

STUDIENARBEIT

Neuroevolution in Roboterschwärmen

VON

Benedikt Müller

**Institut für Angewandte Informatik
und Formale Beschreibungsverfahren
Karlsruher Institut für Technologie**

**Referent: Prof. Dr. Hartmut Schmeck
Betreuer: Dipl. Inf. Lukas König**

Karlsruhe, 28.7.2010

Hiermit bestätige ich, die vorliegende Arbeit selbständig
durchgeführt und keine anderen als die angegebenen
Literaturhilfsmittel verwendet zu haben.

Karlsruhe, 28.7.2010

Benedikt Müller

Inhaltsverzeichnis

1	Einführung	1
2	Grundlagen	3
2.1	Evolutionäre Algorithmen	3
2.2	Neuronale Netze	4
2.3	No-Free-Lunch-Theoreme und Fitnessraum	4
2.4	Koevolution	7
3	Verwandte Arbeiten	9
3.1	Evolutionäre Robotik	9
3.2	NEAT	10
4	Neuronale Architektur	12
4.1	Aufbau	12
4.2	Genetische Operatoren	12
5	Evolution im Putzscenario	16
5.1	Szenariobeschreibung	16
5.2	Schwärme als Individuen	16
5.3	Implementierung	17
5.4	Trajektorienbilder	18
5.5	Ergebnisse	21
6	Koevolution im Fußballscenario	24
6.1	Szenariobeschreibung	24
6.2	Evolution	25
6.3	Implementierung	26
6.4	Fitnessentwicklung	28
6.5	Ergebnisse	29
7	Fazit und Ausblick	33
7.1	Zusammenfassung	33
7.2	Ausblick	33
8	Anhang: Spielszenen	38

1 Einführung

Als Teilgebiet des Organic Computing stehen auch in der Evolutionären Robotik die Self-X-Eigenschaften im Mittelpunkt. Insbesondere die Selbstorganisation, Selbstkonfiguration und Selbstoptimierung werden im Folgenden untersucht. Diese Arbeit beschäftigt sich mit Roboterschwärmen, die durch Evolution lernen. Die Schwärme bestehen aus vielen kleinen Robotern mit begrenzter Sensorik und begrenzten Fähigkeiten. Da die Schwärme oft dezentral agieren, muss jeder Roboter auf Grund von lokalen Informationen seine Entscheidungen treffen. Die Natur führt unter anderem am Beispiel von Hymenopterenstaaten (Ameisen, Bienen) eindrucksvoll vor, dass aus dem Zusammenspiel einzelner wenig intelligenter Akteure mit begrenztem Einfluss ein sehr wirkungsvolles und effizientes globales Verhalten des Schwarms entstehen kann. Dieser Effekt, dass aus einfachen Komponenten im Zusammenspiel ein komplexeres Gesamtsystem entsteht, heißt Emergenz.

Eine definierende Eigenschaft von Emergenz ist, dass das globale Verhalten aus dem Verhalten der Einzelakteure nicht unmittelbar erkennbar ist. Für einen Menschen ist es daher schwierig, emergente Systeme zu entwerfen, die bestimmte Aufgaben erfüllen sollen. Es ist schwer abschätzbar, welches globale Verhalten aus dem lokalen Verhalten von einfachen Akteuren emergiert. Könnte der Roboterschwarm sich hingegen selbst optimieren, z.B. indem er eigenständig lernt, wie ein Problem zu lösen ist, würde dem menschlichen Designer Arbeit abgenommen werden. Evolutionäre Algorithmen stellen eine Möglichkeit dar, dieses Lernen zu erreichen und sollen in dieser Arbeit im Kontext von Schwarmrobotern betrachtet werden.

Die Schwärme müssen sich dazu in zwei simulierten Szenarien beweisen. Im Putzscenario sollen in begrenzter Zeit möglichst viele Stellen im Testgelände von den Robotern besucht werden. Das Roboterfußball-Szenario erfordert die gezielte Interaktion mit dem Ball, der gegen den Widerstand der anderen Mannschaft ins Tor bewegt werden muss. Die Roboter werden von künstlichen neuronalen Netzen gesteuert, die durch einen evolutionären Algorithmus lernen. Das Lernen geschieht zentral gesteuert, die Roboter selbst agieren weitgehend dezentral, auf Basis lokaler Sensorinformationen.

2 Grundlagen

2.1 Evolutionäre Algorithmen

Der Begriff „Evolutionärer Algorithmus“ steht heutzutage für eine ganze Kategorie von Algorithmen, die auf der iterativen Verbesserung von Lösungen für ein Optimierungsproblem beruhen. Vorbild ist dabei der Prozess in der Natur, der durch zufällige Änderungen („Mutationen“) Varianten von bestehenden Individuen erzeugt und die besten davon wieder als Schablone („Eltern“) für neue variierte Individuen („Kinder“) verwendet. Die Individuen, die sich erfolgreicher fortpflanzen, verdrängen dabei diejenigen, die sich weniger erfolgreich fortpflanzen. Diese Evolution hat in der Natur die komplexesten bekannten Systeme erschaffen – von hochangepassten Einzellern bis hin zu multizellulären Verbänden mit stark hierarchischen Organisationsstrukturen.

Die von der Evolution inspirierten Optimierungsverfahren sind sehr stark abstrahiert und müssen in viel kleinerem Maßstab als die Natur arbeiten. Üblich ist die Verwendung einer Fitnessfunktion, die den Lösungskandidaten eine reelle Zahl als Bewertung zuordnet, je nach ihrer Eignung. Besonders geeignete Kandidaten werden dann zur Nachkommengenerierung ausgewählt. In der Natur gibt es keine explizite Fitnessfunktion, neben vielen anderen Kriterien ist das eigene Überleben für den Fortpflanzungserfolg am wichtigsten. Außerdem löst die natürliche Evolution nicht zwangsläufig Optimierungsprobleme, der Grundmechanismus lautet „was sich fortpflanzt, hat Bestand“. Nicht immer ist der bestangepasste Organismus derjenige, auf den dies zutrifft. Das Ziel, auf das hin optimiert werden könnte, ändert sich in der Natur auch ständig.

Eine Stärke von evolutionären Algorithmen ist, dass sie in vielen Bereichen anwendbar sind, in denen kein anderes Optimierungsverfahren bekannt ist. Durch ihre iterative Vorgehensweise können sie auch dort eingesetzt werden, wo sich die Zielvorgaben ändern, sie können die Lösung dann an die veränderten Bedingungen anpassen. Voraussetzung für ihren Einsatz ist, dass Lösungskandidaten für das Optimierungsproblem einfach und genau bewertet werden können. Die Lösungskandidaten müssen in einer Form vorliegen, die einen Mutationsoperator unterstützt, der aus einem gegebenen Kandidaten eine Variation erzeugt. Oft wird außerdem ein Kreuzungsoperator verwendet, der aus 2 Kandidaten einen neuen erzeugt, der eine Mischung von beiden „Eltern“ darstellt. In dieser Arbeit dienen künstliche neuronale Netze als Kandidaten. Ihre Bewertung ergibt sich aus der Interaktion von Robotern, die sie steuern, mit ihrer Umgebung. Ob sich das künstliche neuronale Netz fortpflanzen kann, hängt davon ab, wie gut die Roboter die Aufgabe im Vergleich zu anderen Robotern erfüllen.

2.2 Neuronale Netze

Als Controllerrepräsentation wurden in dieser Arbeit künstliche neuronale Netze (im Folgenden neuronale Netze genannt) gewählt. Neuronale Netze haben wichtige theoretische Eigenschaften, die sie im Vergleich zu anderen Controllerrepräsentationen herausheben. Vorwärtsgerichtete neuronale Netze sind universelle Approximatoren [Cyb89], sie eignen sich dadurch vor allem zur Repräsentation gedächtnisloser Funktionen. Rekurrente neuronale Netze sind turing-vollständig [Hyo96], sie eignen sich also zur Repräsentation von Programmen. Da die Roboter Steuerungsprogramme brauchen und in vielen Szenarien ein Gedächtnis von Vorteil sein kann, verwendet diese Arbeit rekurrente neuronale Netze als Steuerungsprogramme für die Roboter.

Ein neuronales Netz ist definiert durch seine Topologie, also die Anzahl und Verknüpfung der Neuronen, und durch seine Gewichte, wobei typischerweise ein Gewicht pro Verbindung vorhanden ist. Die bekannteste Architektur für neuronale Netze ist das Mehrschichtige Perzeptron [Ros88]. Die Neuronen sind dabei in Schichten angeordnet und erhalten ihre Eingaben nur von Neuronen der direkten Vorgängerschicht. Das Netz ist also vorwärtsgerichtet, die Schichten haben eine definierte Reihenfolge, beginnend bei der Eingabeschicht über die verdeckten Schichten bis zur Ausgabeschicht. Als Trainingsverfahren ist „backpropagation of error“ verbreitet [RHW88]. Dieses Verfahren arbeitet mit Trainingsdaten, die aus Eingabedaten und gewünschten Ausgabedaten bestehen. Es führt einen Gradientenabstieg durch um den Fehler zu minimieren.

2.3 No-Free-Lunch-Theoreme und Fitnessraum

Da ein evolutionärer Algorithmus ein Optimierungsverfahren ist, lohnt es sich, zu betrachten unter welchen Bedingungen ein solches erfolgreich arbeiten kann. Die No-Free-Lunch-Theoreme von Wolpert und Macready [WM97] zeigen eine Schranke für Optimierungsverfahren auf: Wenn man die Menge aller Probleme betrachtet, arbeiten alle Optimierungsverfahren gleich gut oder schlecht. Vorbedingung hierbei ist, dass die Optimierungsverfahren einen Punkt im Fitnessraum nur ein einziges mal evaluieren. Unter den Bedingungen der NFL-Theoreme arbeitet eine zufällige Suche genauso gut wie ein evolutionärer Algorithmus oder Gradientenaufstieg. Selbst Algorithmen, die absichtlich intuitiv „schlechte“ Entscheidungen treffen (ein genetischer Algorithmus, der versucht die Fitness zu minimieren) arbeiten genauso gut. Dies liegt daran, dass in der „Menge aller Probleme“ viele konstruierte und sehr exotische Fitnessräume enthalten sind. Ableiten lässt sich aus den NFL-Theoremen auch, dass ein Algorithmus, der auf einer Teilmenge besser

arbeitet als ein anderer Algorithmus, auf der Komplementmenge schlechter arbeiten muss als der andere Algorithmus. Der Durchschnitt muss immer gleich sein. Ein universeller Algorithmus, der eine zufällige Suche immer schlagen kann, existiert also nicht.

In der Praxis ist es aber nicht notwendig einen universellen Optimierungsalgorithmus zu finden, sondern nur einen in der Praxis nützlichen. Nimmt man in Kauf, dass ein Optimierungsverfahren bei uninteressanten Problemen schlechter arbeitet, kann man bei praxisrelevanteren Problemen eine bessere Performance als eine zufällige Suche erreichen. Auch in einer Teilmenge der „Menge aller Probleme“ können die NFL-Theoreme jedoch gelten. Eine notwendige und hinreichende Bedingung hierfür ist, dass die Problemmenge abgeschlossen unter Permutation ist. Gemeint ist hier die Zuordnung von Individuen aus dem Fitnessraum zu ihrer Fitness.

Gegeben sei ein Problem P aus der Menge M . Wenn nun die Zuordnung der Individuen in P zufällig permutiert werden kann und das resultierende Problem \bar{P} ebenfalls in M liegt, dann ist M abgeschlossen unter Permutation. Die NFL-Theoreme gelten hier, die Suche nach einem Optimierungsverfahren, das für die ganze Problemmenge M überdurchschnittlich gut ist, ist zwecklos. In der Praxis kann dieses NFL-Ergebnis jedoch oft entkräftet werden, da man oft Probleme aus einer Menge lösen will, die nicht abgeschlossen unter Permutation ist.

Das zeigt ein wichtiges Ziel der Forschung an evolutionären Algorithmen und Optimierungsverfahren im Allgemeinen: Die Problemmenge muss präzise genug definiert sein, damit sich besondere Eigenschaften herausheben lassen, die der Algorithmus ausnutzen kann. Da die Bedingungen der NFL-Theoreme oft nicht erfüllt sind, wurde die Wichtigkeit der NFL-Ergebnisse relativiert („no free lunch is no big deal“ [Cul96]). Es ist jedoch Vorsicht geboten, da auch schwächere Versionen der NFL-Theoreme gelten können („almost no free lunch“ [DJW97]).

Zur Veranschaulichung werden im Folgenden nun zwei Problemklassen beschrieben, eine davon ist abgeschlossen unter Permutation (Nadel im Heuhaufen) und eine ist nicht abgeschlossen unter Permutation (Schiffe versenken) (Abbildung 1). Gesucht ist eine Strategie, die in einem Feld der Größe 10×10 möglichst schnell möglichst viele schwarze Kästchen findet. Im Fall des Nadel-im-Heuhaufen-Problems ist jeder Suchalgorithmus gleich gut, da der Suchraum strukturlos ist, hier gelten die NFL-Theoreme. Bei der Schiffe-Versenken-Anordnung liegen hingegen schwarze Punkte häufiger in der Nähe von anderen schwarzen Punkten. Der Raum hat Struktur, die von einem Suchalgorithmus genutzt werden kann.

Thomas Miconi [Mic07] schlägt zur Untersuchung, ob darwinsche Evolution stattfindet, die Messung der „Fitness Transmission“ vor. Mit Fitness

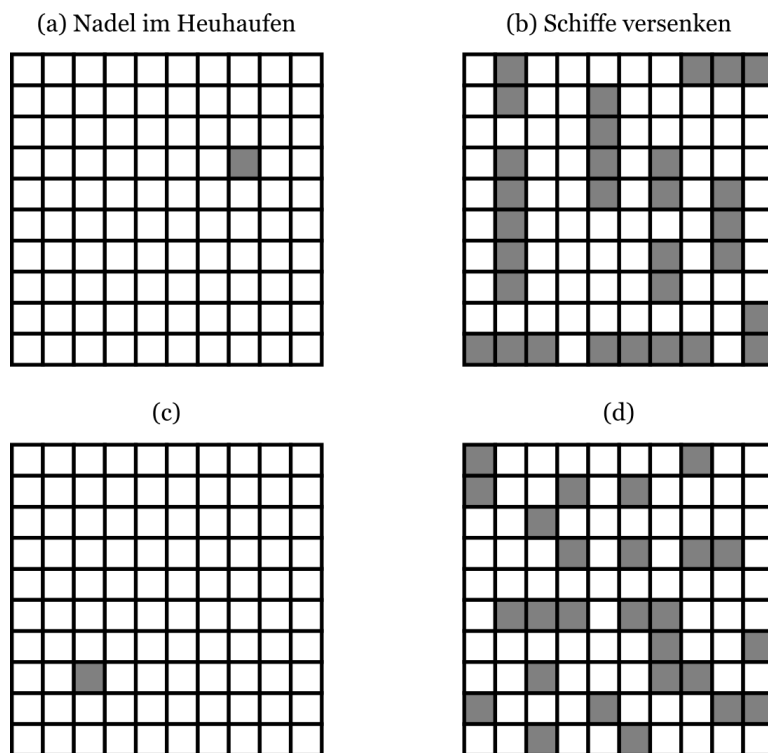


Abbildung 1: (a+c) Das Nadel-im-Heuhaufen-Problem ist abgeschlossen unter Permutation. (b+d) Die zufällige Permutation der Fitnesszuordnung des Schiffe-Versenken-Problems ergibt hingegen kein gültiges Schiffe-Versenken-Problem.

Transmission ist die Korrelation zwischen Eltern- und Kind-Fitness gemeint. Evolutionäre Algorithmen stützen sich auf die Annahme, dass im Fitnessraum nahe beieinander liegende Individuen auch ähnliche Fitnesswerte besitzen. Aus diesem Grund suchen sie die Stellen des Fitnessraums gründlicher ab, in der sich Individuen mit besonders hoher Fitness befinden. Eine zufällige Permutation der Zuordnung von Individuum zu Fitness würde diesen Zusammenhang zerstören, die Struktur des Fitnessraums geht verloren und wird durch Rauschen ersetzt. In so einem strukturlosen Raum ist es nicht überraschend, dass alle Optimierungsverfahren gleich schlecht arbeiten.

2.4 Koevolution

Koevolution bezeichnet eine Folge gegenseitiger Adaptionen von Individuen, die miteinander konkurrieren. Ein Resultat ist die Red-Queen-Hypothese [vV73], die besagt, dass ein Organismus sich ständig anpassen muss, um seine Nische weiterhin erfolgreich besetzen zu können, da sich konkurrierende Organismen ebenfalls ständig an seine Schwächen anpassen. Im Gegensatz zu klassischen evolutionären Algorithmen hat die Koevolution keine feste Fitnesslandschaft. Welche Konfigurationen positiv ist und welche schlecht hängt von der Konfiguration der gegenwärtigen Konkurrenten ab. Die Fitnesslandschaft aus Sicht eines Individuums ändert sich also ständig, da die eigene Fitness zum großen Teil von den Eigenschaften anderer Individuen abhängt, die gleichzeitig eine Evolution durchlaufen. Ein Effekt, der dabei auftreten kann, ist das gegenseitige Wettrüsten der Konkurrenten, z.B. zwischen Parasit und Wirt. Diese „Arms races“ können in bestimmten Fällen die Evolution antreiben, komplexere Lösungen zu entwickeln [DK79].

Bei simulierter Evolution tritt Koevolution vor allem bei Artificial-Life-Simulationen auf, z.B. in Jäger-Beute-Szenarien. Koevolution wurde aber auch schon als Erweiterung von klassischen evolutionären Algorithmen zur Lösung nicht-koevolutionärer Optimierungsprobleme verwendet ([Hil90] und [JP96]). Verfahren, die die Fitness eines Individuums in Bezug zu anderen Individuen setzen, z.B. um Innovationen zu schützen (wie bei NEAT [SM02]) oder große Gruppen sehr ähnlicher Individuen zu bestrafen, können bereits als einfache Form der Koevolution aufgefasst werden. Der Fitnessraum bleibt hier ebenfalls nicht mehr fest, sondern das Ziel, auf das hin optimiert wird, verändert sich aus Sicht des Individuums, da sich andere Individuen verändern. Dieses sich verändernde Ziel kann dazu führen, dass eine frühzeitige Konvergenz des genetischen Algorithmus vermieden wird und die Evolution beschleunigt wird [KNA07]. In dieser Studienarbeit wird Koevolution am Beispiel eines Fußballszenarios betrachtet.

Arms races treten in der Koevolution allerdings nicht zwangsläufig auf.

Typisch ist eine wechselseitige zyklische Adaption, die nach dem Schere-Stein-Papier-Prinzip abläuft. Dabei wechseln die Konkurrenten zwischen einer kleinen Anzahl verschiedener einfacher Strategien hin und her, je nachdem, welche Strategie der Gegenspieler gerade verfolgt. Dies führt dazu, dass die Evolution zwar in Bewegung bleibt, aber dennoch in einem kleinen Bereich des Suchraumes verharrt. Komplexes, besonders ausgefeiltes Verhalten wird so nicht gefördert. Der Grund für diesen Schere-Stein-Papier-Effekt liegt in der Intransitivität der „besser als“-Relation bei Koevolution [Mic09]. Wenn Individuum Schere das Individuum Papier schlägt und Papier das Individuum Stein schlägt, dann bedeutet das nicht, dass Individuum Schere zwangsläufig auch Individuum Stein schlägt. Diese Intransitivität erschwert es, die Fitnessentwicklung in der Koevolution zu messen. Möchte man ein Individuum bewerten, so ist es nach Miconi nur möglich, entweder den historischen Fortschritt zu messen (im Vergleich zu früheren in der Koevolution aufgetretenen Individuen) oder den lokalen Fortschritt (im Vergleich zu den anderen Individuen, die aktuell in der Population sind). Der in aller Regel erwünschte globale Fortschritt (Wie gut ist ein Individuum im Vergleich zu allen möglichen Individuen) ist praktisch nicht messbar.

3 Verwandte Arbeiten

3.1 Evolutionäre Robotik

Auf dem Gebiet der evolutionären Robotik sind die Arbeiten von Floreano und Nolfi grundlegend. Sie führten Experimente mit realen Robotern durch und verwendeten zur Steuerung neuronale Netze. Die verwendeten Roboter (Typ Khepera) haben zwei einzeln ansteuerbare Räder und sind im einfachsten Fall mit Infrarot-Distanzsensoren ausgestattet. Die Aufgaben der Roboter spiegeln sich in verschiedenen Fitnessfunktionen wieder. Eine grundlegende Aufgabe ist es, sich zu bewegen und dabei Kollisionen zu vermeiden. Diese Collision Avoidance wurde dabei durch eine Belohnung der synchronen Vorwärtsbewegung beider Räder und eine Bestrafung von Kollisionen realisiert.

In den Arbeiten von König und Schmeck, die die Grundlagen für diese Studienarbeit schaffen, sind deterministische endliche Automaten die Steuerungsprogramme der Roboter. Automaten haben den Vorteil gegenüber neuronalen Netzen, dass sie keine Black-Box sind, sondern leichter menschenlesbar sind. Die Verhaltensweisen, die in vielen Experimenten der evolutionären Robotik erzielt wurden, lassen sich auch mit endlichen Automaten realisieren. Die Turing-Vollständigkeit von rekurrenten neuronalen Netzen ist dafür nicht erforderlich. Die Analyse der Automaten kann dabei das Verhalten der Roboter deutlich einfacher erklären, als es bei den schwer lesbaren neuronalen Netzen der Fall wäre. Ein Roboterverhalten, das gut verstanden ist, lässt sich durch Menschen auch einfacher verändern und zum Beispiel von simulierten auf reale Roboter übertragen. Der Reality Gap kann somit durch eine Ingenieursleistung überwunden werden, während das gleiche Unterfangen bei einem undurchschaubaren neuronalen Netz viel schwieriger ist. Wenn man den Verhaltensautomaten der Roboter vollständig analysiert und versteht, kann man dem Verhalten auch vertrauen und es modifizieren. Die Automaten erreichen somit, sobald sie verstanden sind, den gleichen Status wie eine vom Menschen entwickelte Lösung, was die Vertrauenswürdigkeit und Veränderbarkeit betrifft.

Die Tatsache, dass die Automaten selbst nicht turing-vollständig sind, muss in Roboterschwärmen kein Nachteil sein. Das emergente Verhalten des Schwarms könnte durchaus höherwertige Berechnungen durchführen als die Einzelroboter. Ein Beispiel für ein System, das turing-vollständig ist, aber aus sehr simplen Einzeleinheiten besteht, ist der Zellularautomat „Regel 30“ [Wol83]. Eine einzelne Zelle reagiert nur regelbasiert auf den Zustand ihrer zwei Nachbarn und ihren eigenen. Die Funktionalität der Zelle ist problemlos durch einen deterministischen endlichen Automaten darstellbar. Im Zusammenspiel vieler Zellen ist das System aber turing-vollständig.

In dieser Studienarbeit werden statt Automaten neuronale Netze verwendet. Der Grund dafür liegt neben ihren starken theoretischen Eigenschaften auch in ihrer einfachen Implementierung und der bereits erforschten Kombination mit evolutionären Algorithmen.

3.2 NEAT

NEAT¹ ist ein Verfahren zur Neuroevolution, das sich gegenüber früheren Verfahren dadurch auszeichnet, dass es sowohl die Topologie als auch die Gewichte des neuronalen Netzes gleichzeitig entwickelt. Die Evolution startet dabei mit minimalen neuronalen Netzen, die komplexifiziert werden, indem neue Neuronen entlang bestehender Synapsen eingefügt werden. NEAT spezifiziert eine Architektur des neuronalen Netzes, die der in dieser Arbeit verwendeten stark ähnelt. Es wird ebenfalls kein mehrschichtiges Perzeptron verwendet, sondern es sind beliebige Verbindungen zwischen Neuronen möglich. Neben der Architektur beschreibt NEAT auch die genetischen Operatoren und einen evolutionären Algorithmus, der besonderes Augenmerk auf den Schutz von Innovationen legt. Im Unterschied zu dieser Arbeit fügt NEAT neue Neuronen verbunden ein: Dabei löst es eine bestehende Synapse auf, fügt ein neues Neuron ein und verbindet es mit 2 Synapsen mit den ursprünglich verbundenen 2 Neuronen. Diese Art des Einfügens soll neue Neuronen direkt in die Funktionalität integrieren, schränkt aber den Mutationsoperator auch ein, da die neuen Neuronen bereits eine vorgegebene Funktionalität haben, sie müssen die vorher vorhandene Synapse schließlich ersetzen.

Alle Synapsen, die durch Mutation neu entstehen, erhalten bei NEAT eine eindeutige Innovationsnummer, die zusammen mit der Synapse vererbt wird und auch bei Veränderungen des Synapsengewichts unverändert bleibt. Diese Nummer wird für den Kreuzungsoperator und zur Nischenbildung in der Evolution verwendet. Der Kreuzungsoperator ahmt mit ihrer Hilfe die homologe Rekombination der DNA² in der Natur nach. Über die Nummern können zusammengehörige Sequenzen der beiden Genome erkannt und relativ zueinander gleich ausgerichtet werden. Somit wird es wahrscheinlicher, dass das resultierende Genom nach der Kreuzung noch ähnlich funktioniert wie die Elterngenome.

Der Lernalgorithmus berechnet die Ähnlichkeit zwischen zwei Genomen rein über die Innovationsnummern. Das ist eine Alternative zu Ansätzen, die beispielsweise den euklidischen Abstand zwischen den Gewichtsvektoren der Netze berechnen. Mit der Hilfe dieses Ähnlichkeitsmaßes werden Innovatio-

¹Neuroevolution of augmenting topologies

²Desoxyribonukleinsäure

nen in eigenen Nischen geschützt und können eine Zeit lang eine Evolution durchlaufen, bevor sie sich gegenüber den anderen Individuen beweisen müssen. Der Hintergrund ist der, dass größere Veränderungen immer erst eine Zeit brauchen, bis sie voll funktionsfähig sind und solange geschützt werden müssen, da die Evolution sie sonst rasch eliminiert und im lokalen Optimum sitzen bleibt.

4 Neuronale Architektur

4.1 Aufbau

Da in der evolutionären Robotik Trainingsdaten nicht direkt zur Verfügung stehen und das Backpropagation-Verfahren sehr stark zu lokalen Optima konvergiert, wird in dieser Arbeit ein anderes Verfahren verwendet. Als grundlegende Architektur dient die Cascade-Correlation-Architektur von Fahlman ([FL90] und [Fah90]), da sie beliebige Verbindungen zwischen Neuronen zulässt und eine Komplexifizierung wie in NEAT unterstützt. Fahlmans Lernalgorithmus wird nicht verwendet, stattdessen kommen evolutionäre Algorithmen zum Einsatz.

In der Cascade-Correlation-Architektur kann jedes Neuron mit jedem beliebigen Vorgänger- und Nachfolgerneuron verbunden sein. Die Konnektivität lässt sich dann als Adjazenzmatrix darstellen, wie in Abbildung 2 angedeutet. Die Neuronen am Anfang des Netzes haben eine Sonderstellung, das erste Neuron ist das Schwellwertneuron, danach folgen Eingabeneuronen. Am Ende des Netzes stehen die Ausgabeneuronen. Für die Verwendung zur Steuerung der Roboter liegen an den Eingabeneuronen Sensorwerte an und die Ausgabeneuronen liefern zum Beispiel Geschwindigkeitswerte zur Ansteuerung der Räder (siehe Abbildung 3).

Die Adjazenzmatrix kostet oft zu viel Speicherplatz, da ihr Platzbedarf mit $O(n^2)$ steigt (n ist die Anzahl der Neuronen). Deswegen wird stattdessen eine Adjazenzzliste gespeichert (siehe Abbildung 6). Hier ist der Speicherbedarf nur noch in $O(n + s)$ (s ist die Anzahl der Synapsen). Da die Anzahl möglicher Synapsen quadratisch mit der Anzahl der Neuronen steigt, spart dies nur Platz, wenn die Adjazenzmatrix dünn besetzt ist, also nur wenige Synapsen tatsächlich vorhanden sind.

4.2 Genetische Operatoren

- **Neuron einfügen:** Die Komplexifizierung besteht darin, dass neue Neuronen und Synapsen eingefügt werden und die neuronalen Netze somit wachsen. Im Gegensatz zu Fahlmans Algorithmus, der neue Neuronen nur am Ende des Netzes einfügt, kann der Einfügeoperator neue Neuronen an beliebigen Stellen einfügen (siehe Abbildung 4), nur die Ein- und Ausgabeneuronen bleiben fest an ihrer Position.
- **Synapse einfügen:** In einem Neuron wird eine Synapse zu einem bislang noch nicht mit diesem Neuron verbundenen anderen Neuron geschaffen.

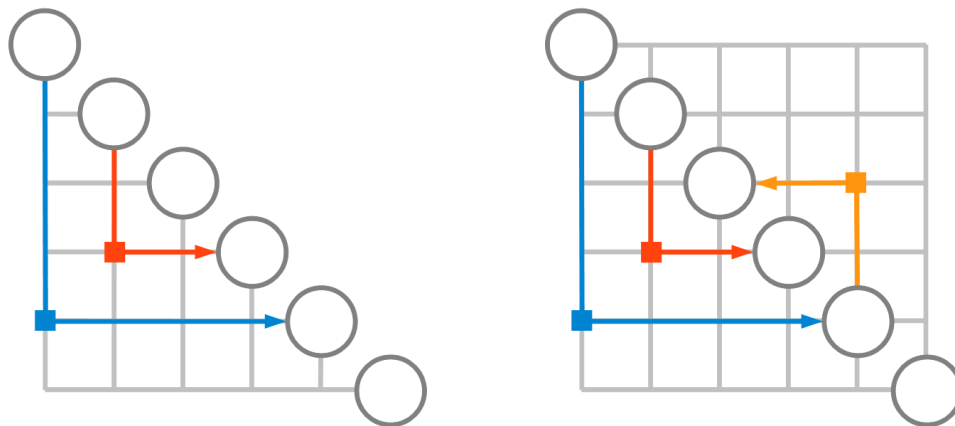


Abbildung 2: Links: Treppenartig aufgebautes vorwärtsgerichtetes Netz. Rechts: Netz mit rekurrenten (rückwärtsgerichteten) Synapsen.

- **Neuron löschen:** Nicht nur eine Komplexifizierung sondern auch eine Vereinfachung ist möglich, indem Neuronen und Synapsen wieder gelöscht werden. Die Synapsen zu den gelöschten Neuronen werden dabei auch gelöscht, die Indizes aller anderen Synapsen werden angepasst (siehe Abbildung 5).
- **Synapse löschen:** Eine Synapse eines Neurons zu einem anderen wird entfernt.
- **Synapsengewicht mutieren:** Das Gewicht einer beliebigen Synapse wird verändert, indem ein normalverteilter Wert addiert wird.

Die Funktionsweise der genetischen Operatoren wurde bei einer Klassifikationsaufgabe getestet. Das Zwei-Spiralen-Problem (siehe Abbildung 7) ist dabei ein häufig verwendetes Benchmarkproblem. Das neuronale Netz bekommt als Eingabe eine x- und eine y-Koordinate und soll klassifizieren, zu welcher der beiden ineinander verschränkten Spiralen der Punkt gehört. Das Problem ist für Gradientenaufstiegsverfahren wie z.B. Backpropagation schwer zu lösen und wurde auch von einem evolutionären Algorithmus mit den hier beschriebenen Operatoren nur vereinzelt gelöst. Die erfolgreichen Lösungen zeigen aber, dass die Operatoren gut funktionieren können und in der Lage sind, komplexe neuronale Netze zu erstellen.

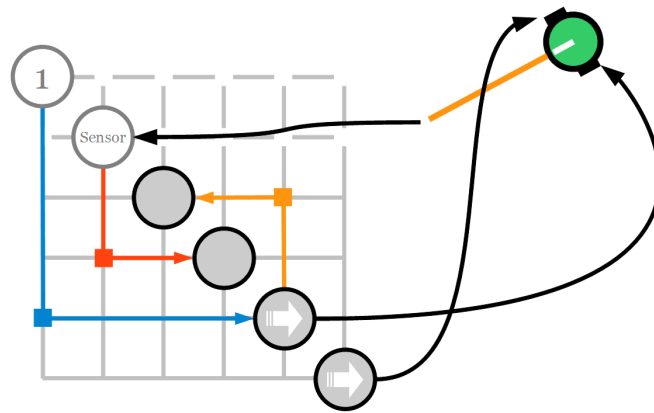


Abbildung 3: Ein-/Ausgabe-Neuronen.

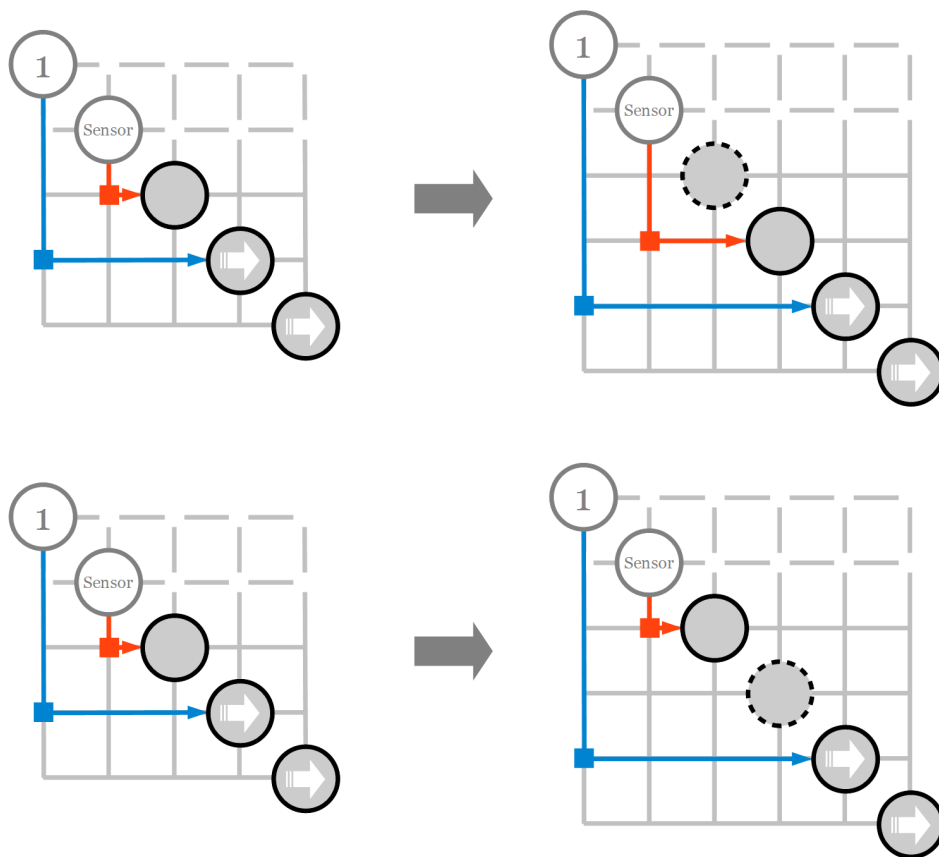


Abbildung 4: Zwei mögliche Einfügepositionen für ein neues Neuron.

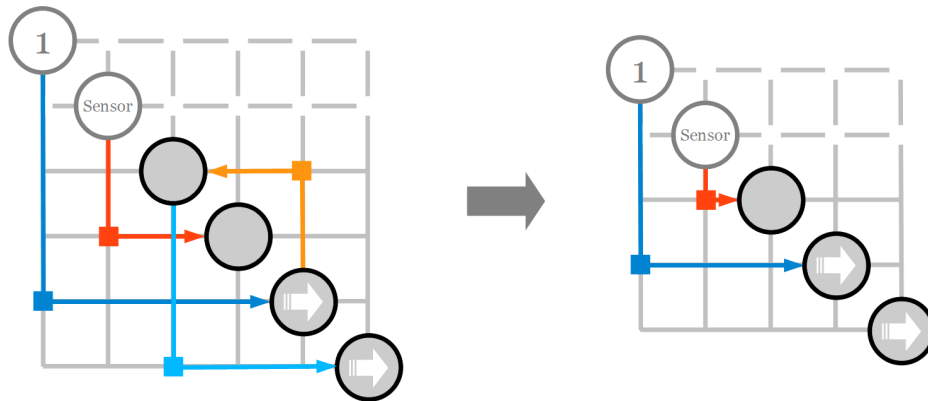


Abbildung 5: Neuron wird gelöscht.

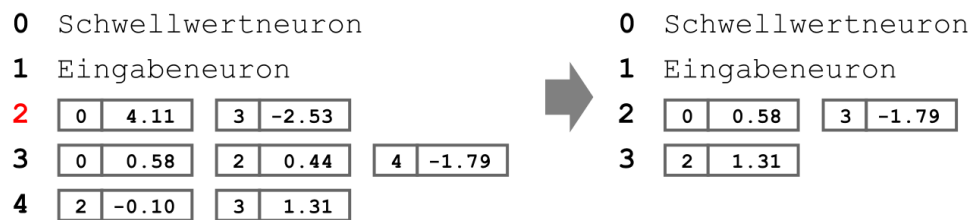


Abbildung 6: Neuronales Netz als Adjazenzliste. Eine Zeile steht für ein Neuron, die Neuronen sind gemäß ihrer Reihenfolge aufsteigend nummeriert. Die Neuronen enthalten die aus Index und Gewicht bestehenden Eingangssynapsen. Im Bild dargestellt ist die Löschung des Neurons „2“. Alle Verbindungen zu Neuron 2 werden gelöscht, alle Indizes werden angepasst.

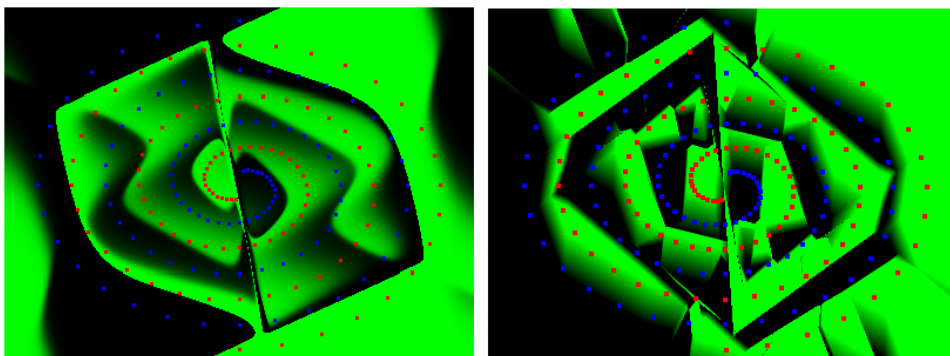


Abbildung 7: Ausgewählte gute Approximationen für das Zwei-Spiralen-Problem. Das Problem diente als Test für die neuronale Architektur ohne rekurrente Synapsen.

5 Evolution im Putzscenario

5.1 Szenariobeschreibung

Das Putzscenario wurde als logische Erweiterung von Collision Avoidance ausgewählt. In die Fitness-Funktion geht dabei eine Bestrafung für Kollisionen ein, aber anstatt die Motoransteuerung zu bewerten, wird die besuchte Fläche belohnt („Reinigung“). Dies gibt den Robotern einen Anreiz, ein möglichst großes Gebiet der Karte zu erkunden, ohne dabei zu kollidieren. Im Fall eines kooperativen Schwarms wird dadurch auch eine Arbeitsteilung und möglicherweise Kommunikation gefördert, da es ein Nachteil ist, wenn mehrere Roboter das gleiche Gebiet reinigen.

Die simulierten Roboter entsprechen von den Bewegungsfähigkeiten her den typischen Schwarmrobotern wie bei Nolfi und Floreano [D.00] oder König und Schmeck [KS08]. Sie haben 2 Räder mit festen Achsen, die sie einzeln ansteuern können. Kurven fahren sie, indem sie beide Räder unterschiedlich schnell bewegen, auf der Stelle drehen ist mit genau entgegengesetzter Ansteuerung möglich. Der Roboterkörper wird als rund angenommen. Ihre Umgebung nehmen die Roboter über exakte Distanzsensoren wahr. Die Roboter sind entweder mit 5 festen Distanzsensoren ausgestattet, die rundherum verteilt ausgerichtet sind oder mit einem einzelnen, den sie frei bewegen können. Da sich das resultierende Verhalten bei beiden Sensorausstattungen kaum unterscheidet, wird im Folgenden meist der einzelne bewegliche Sensor verwendet.

5.2 Schwärme als Individuen

Um die erwähnte Kooperation zu fördern verwendet diese Studienarbeit Schwärme als Individuen in der Evolution. In der Bewertung zählt nur die Gesamtleistung des Schwarms, Altruismus (selbstloses Verhalten einzelner Roboter zum Wohl des Schwarms) wird dadurch gefördert. Jeder Schwarm definiert sich durch ein einzelnes neuronales Netz. Viele solcher Schwärme (z.B. 100) bilden eine Population. Sie werden nacheinander einzeln in der Simulation evaluiert. Dazu wird ein Schwarm aus der Menge der noch nicht evaluierten Schwärme ausgewählt und es wird pro Roboter in der Simulation eine Kopie des neuronalen Netzes des Schwarms erzeugt. Dieses Netz steuert dann den zugehörigen Roboter, die Simulation läuft eine vordefinierte Anzahl von Zyklen lang. Am Ende fließt die addierte Gesamtbewertung aller Roboter in die Fitness des Schwarms ein. Die Evolution läuft über einen einfachen evolutionären Algorithmus mit Truncation Selection. Die 50% schlechtesten Schwärme werden eliminiert. Ihren Platz nehmen Kopien von zufällig ausgewählten Schwärmen aus den besten 50% ein. Diese neuen

Schwärme müssen dann evaluiert werden und bilden zusammen mit ihren Eltern und den anderen erhalten gebliebenen Schwärmen die nächste Generation, aus der wieder die 50% schlechtesten eliminiert werden.

5.3 Implementierung

Hauptziel der Implementierung des Putzzenarios war eine hohe Geschwindigkeit der Simulation. Erreicht wurde dies durch Einschränkungen der Beschaffenheit der Umgebung und der Roboter. Die Umgebung ist eine Kästchenwelt, also rasterartig aus Quadraten aufgebaut, die entweder ausgefüllt sind (Wand) oder frei. Die Roboter sind rund, ihr Durchmesser entspricht der Kantenlänge eines Kästchens. Ein Profiling hat gezeigt, dass der weitaus größte Anteil der Rechenzeit für die Berechnung der Auftreffpunkte der Sensorstrahlen verwendet wird. Hierfür ist vor allem eine schnelle Kollisionsabfrage nötig. Diese wird durch die Einschränkungen der Simulation deutlich vereinfacht. Die Kollision von Roboter und Wänden ist trivial, es muss lediglich in den 4 Feldern, in denen Teile des Roboters sein können, überprüft werden, ob eine Wand vorhanden ist.

Die Kollision von Roboter zu Roboter und Strahl zu Roboter wird über eine Uniform-Grid-Struktur schneller berechnet. Das Uniform Grid ist ein Raster, das eine grobe Repräsentation der dynamischen Szene enthält. Eine Zelle im Raster hat die gleiche Auflösung wie die Landschaft. Jede Zelle im Uniform Grid enthält eine Liste der Roboter, die in diese Zelle hineinragen. Diese Liste wird nach jedem Simulationszyklus aktualisiert. Dadurch muss für einen Roboter nur noch in 4 Gitterzellen überprüft werden, ob andere Roboter vorhanden sind. Nur falls dies positiv ausfällt, muss berechnet werden, ob die Roboter tatsächlich kollidieren.

Für die Strahlberechnung wandert der Strahl das Uniform Grid und das statische Landschaftsgitter entlang und überprüft alle dabei besuchten Zellen auf vorhandene Roboter und Wände. Wenn der Strahl mit einer Wand kollidiert, wird der Auftreffpunkt auf der Wand berechnet. Bei im Uniform Grid vorhandenen Robotern wird überprüft, ob der Strahl den Roboter tatsächlich trifft und dann der Auftreffpunkt auf dem runden Roboter berechnet.

Ein Simulationszyklus ist in folgende Phasen aufgeteilt:

1. **Sensorberechnung:** Die Startpunkte und Winkel der Sensoren werden bestimmt und die Strahlen wandern über die Gitter-Datenstrukturen bis der Auftreffpunkt gefunden wurde. Dieser wird für jeden Sensor gespeichert.
2. **Neuronale Netze propagieren:** Die neuronalen Netze der Roboter

erhalten alle Sensordaten des Roboters als Eingabe und berechnen als Ausgabe die Ansteuerwerte für die Räder des Roboters und den Richtungswert für drehbare Sensoren.

3. **Bewegung:** Alle Roboter berechnen ihre neue Position gemäß ihrer Radmotoransteuerung und setzen ihre Koordinaten auf die neue Position und Rotation. Die vorherige Position und Rotation wird gespeichert.
4. **Kollisionsbestimmung:** Alle Roboter prüfen nun mit Hilfe der Gitterdatenstrukturen, ob sie kollidieren. Alle Roboter, die kollidieren, werden auf ihre frühere Position und Rotation zurückgesetzt.
5. **Uniform Grid aktualisieren:** Für alle Roboter, die nicht kollidieren, wird das Uniform Grid mit ihrer neuen Position aktualisiert: Der Roboter wird aus den Listen der 4 Zellen seiner zuvor besetzten Position gelöscht und in die Listen der neuen 4 Zellen eingetragen.

Ein Simulationszyklus ist gut parallelisierbar, da jeder einzelne Roboter für alle Phasen nur die Informationen aus dem vorherigen Simulationszyklus benötigt. Lediglich die letzte Phase benötigt Synchronisation, da die Listen im Uniform Grid nicht parallel verändert werden können. Im Rahmen dieser Studienarbeit wurde eine Parallelisierung des Simulationszyklus begonnen, aber nicht fertiggestellt. Dabei wurden in jedem Zyklus die Threads neu gestartet (beschleunigt mit einem Thread Pool) und am Ende wieder zusammengeführt (join). Der Overhead hierbei war allerdings deutlich zu hoch und amortisiert sich nur bei sehr vielen Sensorberechnungen. Um diesen Overhead zu reduzieren, ist es sinnvoller, eine Barriere am Ende jedes Zyklus einzuführen und den Zugriff auf das Uniform Grid über feingranulare Sperren zu synchronisieren. Die Threads sollten dabei bis zum Ende der Evaluierung (also mehrere Hunderte oder Tausende von Zyklen lange) laufen, da das Starten und Zusammenführen nicht praktikabel ist. Das Putzszenario wurde somit nur sequentiell durchgeführt, für das Fußballsszenario wurde eine einfacher zu implementierende Parallelisierung auf einer höheren Ebene eingeführt.

5.4 Trajektorienbilder

Abbildung 8 zeigt ein typisches Verhalten, wenn Kollisionen nicht bestraft werden und nur die Reinigungsleistung belohnt wird. Wenn die Landschaft es zulässt, genügt es, einfach konstant Kreise zu fahren, durch die Kollision mit den Wänden wird die Vorwärtsbewegung realisiert und es entsteht automatisch ein Wall-Following (siehe Abbildung 8). Dieses Verhalten ist

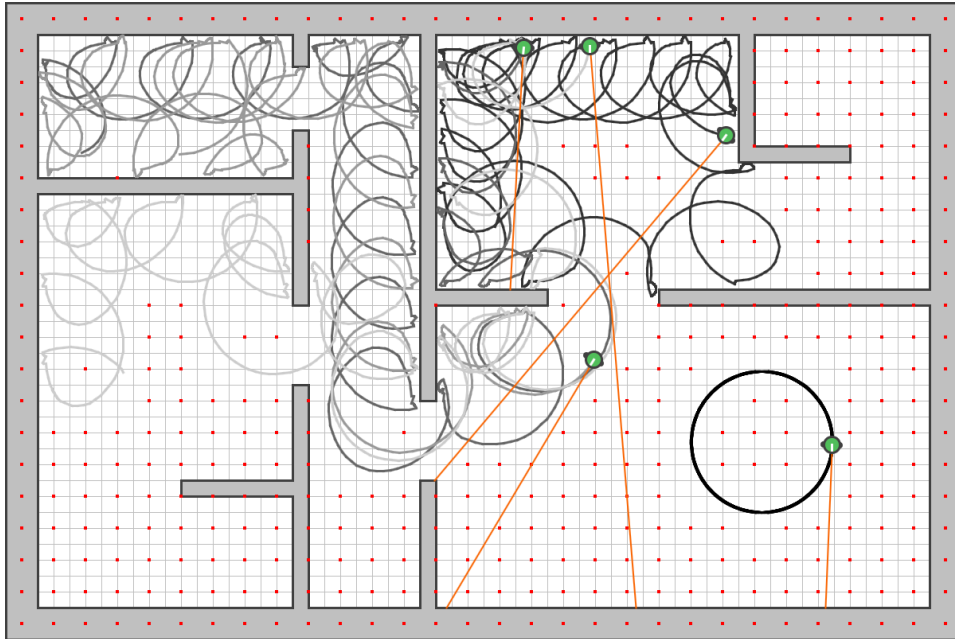


Abbildung 8: Statisches Kreisen, der Sensor wird weitgehend ignoriert. Der Putzerfolg entsteht rein mechanisch durch die Kollisionen zwischen Robotern und Wänden.

ein sehr dominantes lokales Optimum, zu dem die Evolution in aller Regel konvergiert.

Um ein etwas komplexeres Verhalten zu erreichen, ist die Kollisionsbe-
strafung vorteilhaft. Abbildung 9 zeigt ein frühes Resultat in der Evolution.
Die Roboter fahren in diesem Fall modifizierte Kreise. Dadurch bleiben sie
mit Hilfe des Distanzsensors eher in der Mitte der Räume und bewegen sich
langsam vorwärts.

Wenn die Evolution noch eine Weile weiter läuft verändert sich dieses
Verhalten (Abbildung 10). Die Roboter haben sich den Wänden stark ange-
nähert und betreiben ein kollisionsfreies, kreisendes Wall-Following. Dieses
Verhalten funktioniert bei manchen Landschaften sehr gut, da die Robo-
ter viele Teile der Landschaft mit Hilfe der Wände besuchen, es reinigt die
Raummitten aber grundsätzlich nicht. Auch dieses lokale Optimum ist wie-
der so stark, dass es das Resultat praktisch jedes Evolutionslaufes ist.

Um die Entwicklung komplexerer Strategien zu erleichtern, wurden die
Distanzsensoren als nächstes erweitert. Neben dem Distanzwert stellen sie
den Robotern jetzt auch die Information zur Verfügung, wie viele schmutzige
Felder der Strahl bis zur Wand passiert hat. In Abbildung 11 sieht man, dass
die Roboter diese Information nutzen, um eine Reinigung durchzuführen, die

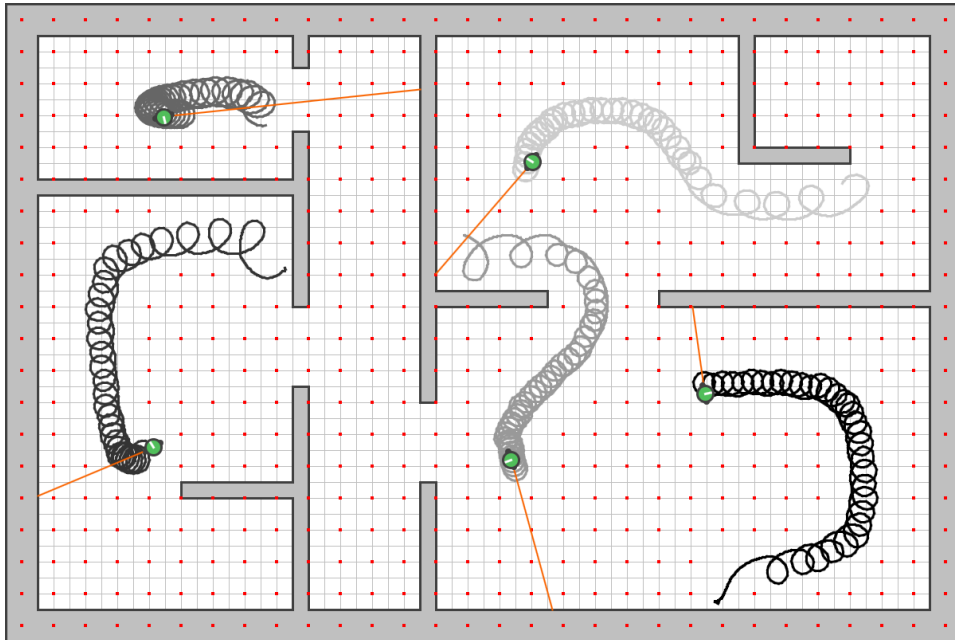


Abbildung 9: Dynamisches Kreisen, die Roboter müssen sich von den Wänden fernhalten und nutzen dafür ihre Distanzsensoren.

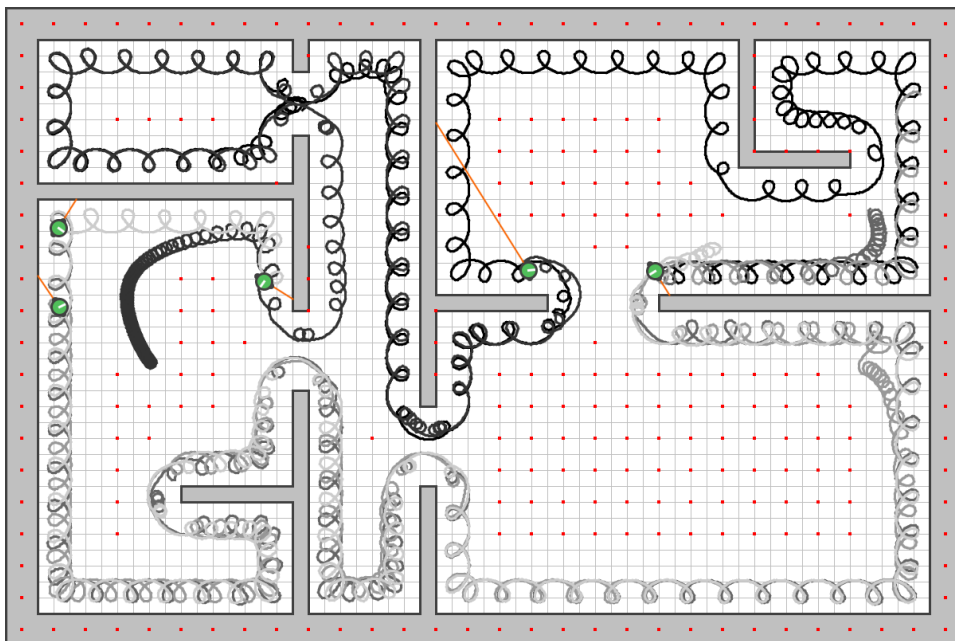


Abbildung 10: Kreisendes Wall-Following.

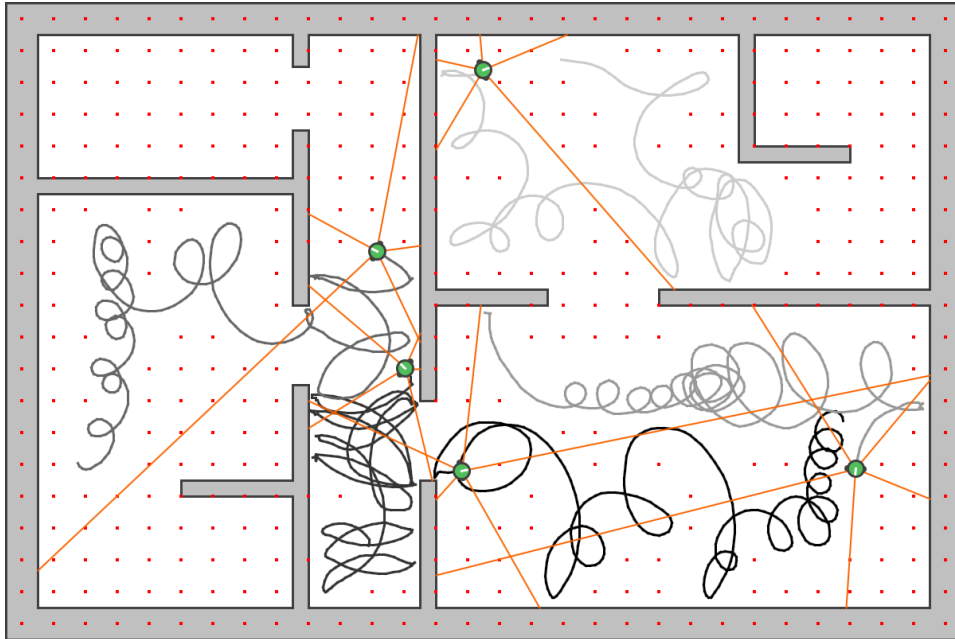


Abbildung 11: Der Distanzsensor wurde hier um Verschmutzungsgrad-Informationen erweitert.

sowohl Raummitten als auch Wände abdeckt.

5.5 Ergebnisse

In der Evolution von neuronalen Netzen für Putzroboterschwärme hat sich vor allem das kreisende Wall-Following als dominante Strategie herausgebildet. Erst durch das Hinzufügen von Schmutzsensoren wurde eine bessere Reinigung erzielt. Sowohl mit als auch ohne Schmutzsensoren verwenden die Strategien aber oft nur minimale neuronale Netze, die keine verdeckten Neuronen benötigen. Falls verdeckte Neuronen vorhanden sind, so führen diese nur zur Feinjustierung des Verhaltens, nicht zu einem Verhalten von völlig anderer Qualität (siehe Abbildung 12). Doch auch mit diesen einfachen Verhaltensweisen wird mit Schmutzsensoren und über längere Zeit hinweg bereits eine sehr umfassende Reinigung erreicht (siehe Abbildung 13). Raum für eine weitere Evolution besteht hier nur noch in einer Effizienzsteigerung und unter Umständen in schwierigeren Umgebungen. Es erschien zweifelhaft, ob das Putzscenario in der beschriebenen Form komplexes Verhalten erfordert und fördert. Deswegen wurde stattdessen für die weiteren Experimente ein Fußball-Szenario ausgewählt.

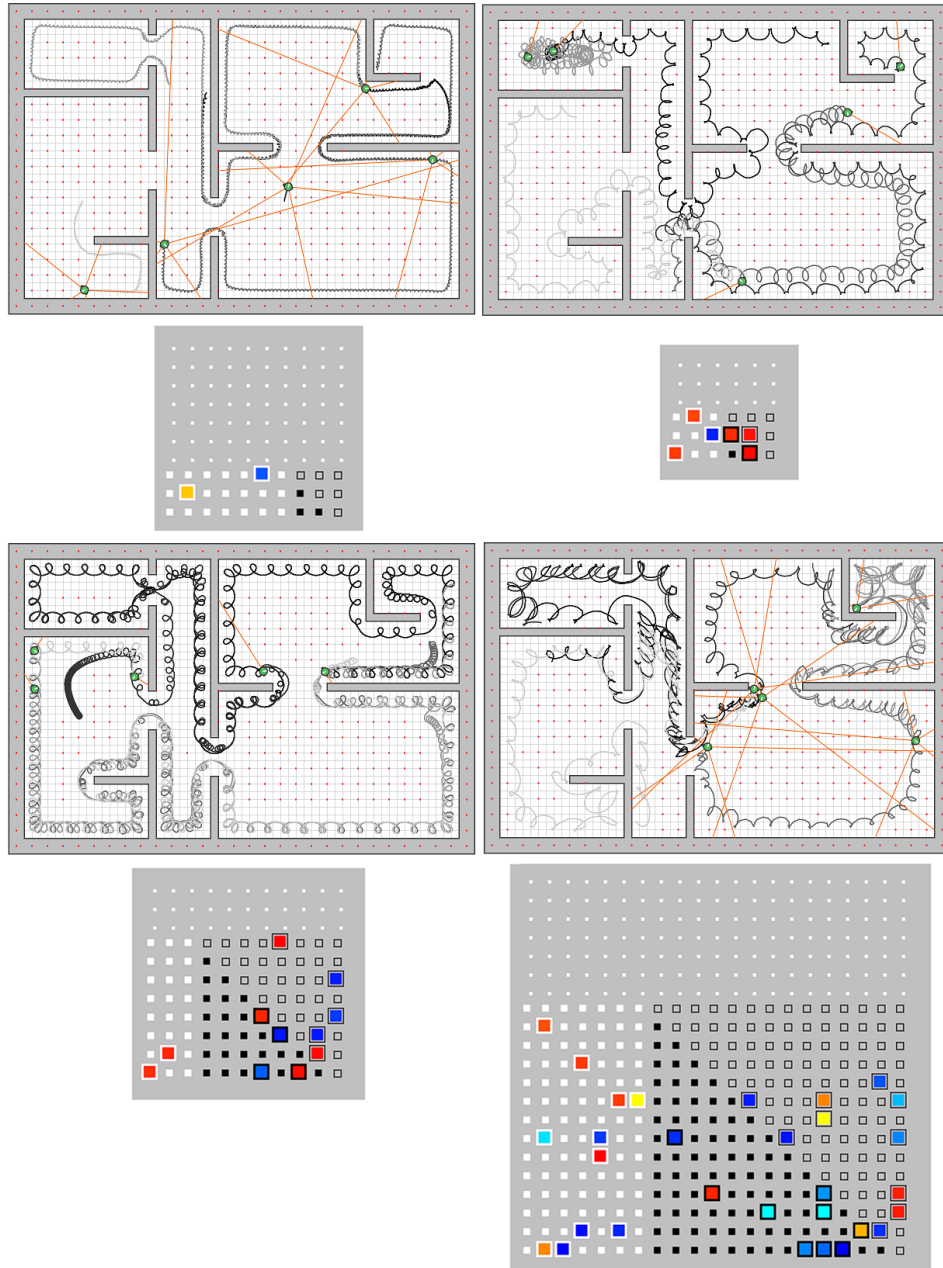


Abbildung 12: Verschiedene Trajektorienbilder und darunter die zugehörigen neuronalen Netze als Hinton-Diagramme. Das Diagramm stellt die Adjazenzmatrix des Netzes da, eine Zeile steht für ein Neuron, farbige Kästchen für Synapsen. Die Darstellung ist hier im Detail nicht wichtig, lediglich die Tatsache, dass die Größe der Matrix und die Anzahl der farbigen Kästchen die Komplexität des neuronalen Netzes ausmachen. Obwohl die neuronalen Netze unterschiedlich viele Neuronen haben, so ist das Verhalten doch sehr ähnlich. Die zusätzliche Komplexität führt nur zu einer Verfeinerung des Verhaltens, nicht zu einer anderen Verhaltenskategorie.

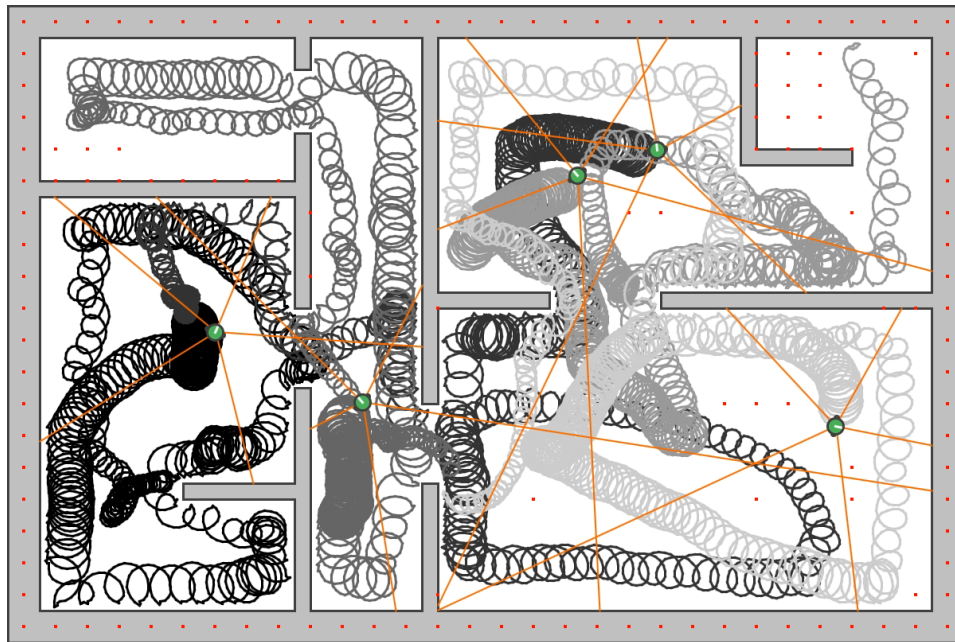


Abbildung 13: Ein Langzeitlauf einer einfachen Strategie mit Schmutzsensoren reinigt große Teile des Testgeländes.

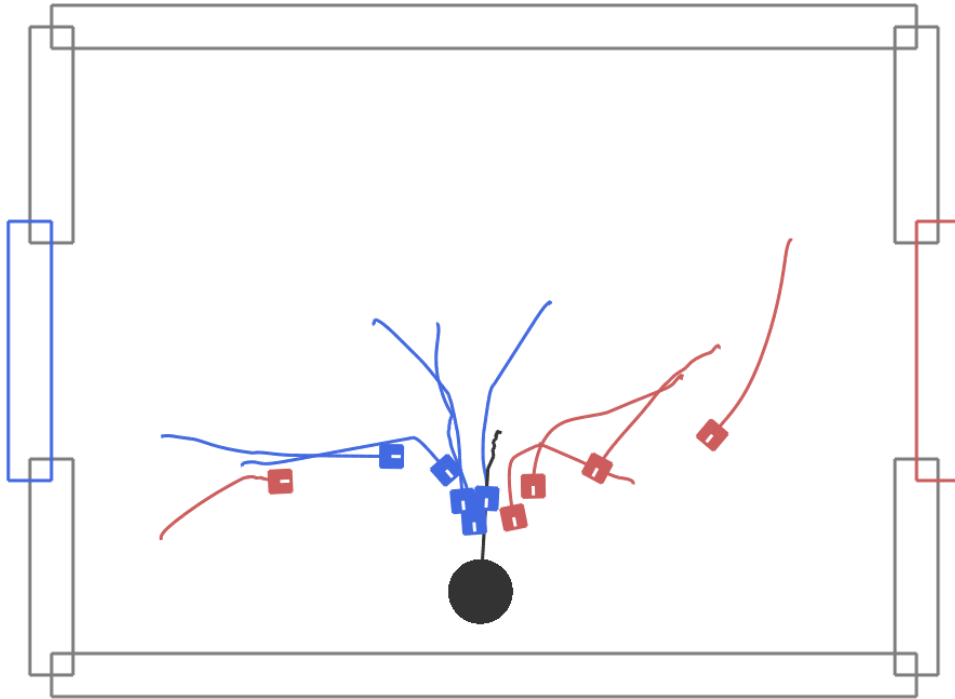


Abbildung 14: Das Spielfeld. Die Fahrspuren der Roboter sind farbig eingezeichnet, die Spur des Balls ist schwarz. Die Roboter haben sich von ihren Startpositionen aus in Richtung Ball gedreht und fahren auf ihn zu.

6 Koevolution im Fußballszenario

6.1 Szenariobeschreibung

Fußballspielen gilt als ein schweres Benchmarkproblem der Künstlichen Intelligenz. Es wurde daher ein Fußballszenario gewählt, um komplexeres Verhalten als im Putzszenario zu fördern. Hierfür wurde ein neues Simulationsprogramm in C++ geschrieben, da hier eine geeignete Physik-Bibliothek zur Verfügung stand (Box2D³). Das Fußballspiel ist stark vereinfacht, die Wände können als Bande genutzt werden, es gibt keine Regeln, außer dass der Ball nach Toren in die Mitte des Spielfeldes zurückgesetzt wird. Es spielen Mannschaften von je 5 Robotern gegeneinander, es gibt keinen Torwart, alle Roboter dürfen auch ins Tor hineinfahren (siehe Abbildung 14). Das Konzept der Schwärme als Individuen wurde für das Fußballszenario übernommen, da es gut zu dem Konzept von Mannschaften passt und die grundlegenden evolutionären Verfahren des Putzszenarios übernommen werden konnten.

Letztendlich ist das Ziel im Fußballszenario das Schießen von Toren, wobei

³<http://www.box2d.org/>

der Ball nach jedem Tor automatisch in die Mitte des Feldes zurückgesetzt wird. Die Evolution benötigt jedoch insbesondere am Anfang eine feiner aufgelöste Staffelung der Fitness. Es sollen auch Individuen belohnt werden, die noch keine Tore schießen, deren Verhalten aber bereits in die richtige Richtung geht. Nolfi und Floreano bezeichnen dies als „bootstrap“ [D.00]. Die Fitnessfunktion setzt sich deshalb aus 3 Komponenten zusammen:

1. **Nähe der Roboter des Schwarms zum Ball.** Diese grundlegende Komponente sorgt dafür, dass die Roboter die Kompetenz entwickeln, den Ball zu erkennen und sich auf ihn zuzubewegen. Sie ist so gewichtet, dass sie die Gesamtfitness um wenige 100 beeinflussen kann.
2. **Nähe des Balls zum gegnerischen Tor.** Diese Fitnesskomponente ist 0, solange die Roboter den Ball nicht berühren. Durch die Berührung wird der Ball angestoßen, in Richtung des eigenen oder gegnerischen Tores. Diese Komponente sorgt dafür, dass die Roboter belohnt werden, wenn sie den Ball in Richtung gegnerisches Tor bewegen. Diese Fitnessfunktion bewegt sich ca. im Bereich bis 1000.
3. **Geschossene Tore - Gegentore.** Diese Funktion zählt letztlich die eigenen Tore positiv und die gegnerischen Tore negativ. Jedes eigene Tor erhöht diese Bewertung um 10000, jedes gegnerische Tor verringert sie um 10000.

6.2 Evolution

Anstelle der Truncation Selection des Putzscenarios verwendet das Fußballscenario Tournament Selection. Um den Zeitaufwand für die Evaluierung möglichst gering zu halten, erhält ein Schwarm nur eine Chance, seine Eignung unter Beweis zu stellen. Es gibt eine einzige Population, die aus typischerweise 80 Schwärmen besteht. Es werden fortwährend 2 Schwärme ausgewählt, diese spielen eine gewisse Zeit lang gegeneinander und der Schwarm, der die höhere Fitness erreicht hat, gewinnt. Er überschreibt das Genom des Unterlegenen mit einer mutierten Version seines eigenen Genoms (Abbildung 15). In der Terminologie von Miconi ist dies ein 1-Strikes-Out-Tournament [MC06].

Stanley und Miikkulainen schlagen für die kompetitive Koevolution einen anderen Experimentaufbau mit zwei getrennten Populationen vor [SM04]. Dieser Aufbau wird im Folgenden auch auf seine Tauglichkeit im Fußballscenario untersucht. Der untere Teil von Abbildung 15 zeigt die beiden getrennten Populationen. Beide werden abwechselnd eine Generation lang evaluiert. Während die eine Population evaluiert wird, dient die andere als Benchmark. Es kommen wieder 1-Strikes-Out-Tournaments zum Einsatz, doch

diesmal spielen die verglichenen 2 Mannschaften nicht gegeneinander, sondern beide jeweils einmal gegen eine zufällig ausgewählte Mannschaft aus der Benchmark-Population. Um eine hohe Fairness zu gewährleisten, werden die Startpositionen der Mannschaften und des Balls zwar zufällig ermittelt, bei beiden Matches wird jedoch eine identische, gespeicherte Startposition verwendet. So kann es zwar vorkommen, dass die Startposition unfair im Vergleich zur Benchmark-Mannschaft ist, aber es interessiert ohnehin nur die relative Fitness zwischen den Tournament-Mannschaften, sodass dies kein Nachteil ist.

Wie im Grundlagenteil/Koevolution beschrieben, ist es bei koevolutionären Szenarien schwierig, die Fitness eines Individuums aussagekräftig zu bestimmen. Um dennoch eine Abschätzung der Fitnessentwicklung zu erhalten, werden die Individuen im Folgenden bezüglich einer Grundkompetenz beurteilt: Fußballspielen bei bewegungslosem Gegner. Damit dies erfolgreich gelingt, müssen viele Strategien entwickelt sein, die auch mit Gegner erforderlich sind. Der Ball muss lokalisiert werden, von der richtigen Seite angestoßen und an dem Gegner vorbei ins gegnerische Tor gebracht werden. Der Ball erscheint dann sofort wieder in der Mitte, woraufhin sich die Roboter reorientieren und wieder auf die richtige Seite des Balles kommen müssen.

Im ersten Schritt wurden die Roboter mit einem beweglichen Distanzsensoren ähnlich wie im Putzscenario ausgestattet. Dieser liefert jetzt neben dem Distanzwert aber auch noch eine andere Information über den Auftreffpunkt des Strahls: den Typ des getroffenen Objekts (Wand, Ball, Roboter aus dem eigenen oder Roboter aus dem gegnerischen Team). Die Roboter können außerdem ihre Farbe ändern. Der Sensorstrahl teilt nur Teammitgliedern den Farbwert des getroffenen Roboters mit. Diese Farbkommunikation wird im Folgenden „Glühwürmchen-Kommunikation“ genannt. Zusätzlich haben die Roboter einen Kompass, der ihnen ihre Rotation relativ zum gegnerischen Tor mitteilt. Dies ist erforderlich, da die Roboter aufgrund des symmetrischen Spielfeldes sonst das eigene nicht vom gegnerischen Tor unterscheiden können.

In einem zweiten Schritt wurde diese Sensorausstattung noch erweitert. Die Roboter haben nun einen zweiten Kompass, der ihnen ihre Rotation im Vergleich zum Mittelpunkt des Balles angibt. Außerdem bekommen sie die globale Information, wie weit sie vom Mittelpunkt des Spielfeldes entfernt sind und wie weit der Ball vom Mittelpunkt des Spielfeldes entfernt ist.

6.3 Implementierung

Die komplette Simulation von der Kollisionsabfrage bis zur Berechnung des Sensorstrahl-Auftreffpunktes übernimmt beim Fußballscenario die Box2D-

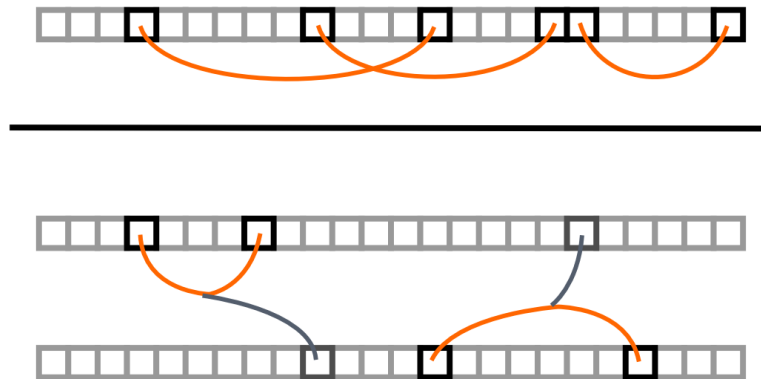


Abbildung 15: Die zwei getesteten Arten der Koevolution. Oberes Diagramm: Eine Einzelpopulation, je 2 Schwärme treten gegeneinander an, der Gewinner überschreibt das Genom des Verlierers mit einer mutierten Kopie seines eigenen Genoms. Unten: Zwei Populationen, die obere steht für die rechte Mannschaft, die untere für die linke Mannschaft auf dem Spielfeld. Für die Evolution werden 2 Mannschaften aus der selben Population ausgewählt, diese treten beide jeweils einmal gegen eine Mannschaft aus der anderen Population an, die als Prüfstein fungiert. Wer von beiden besser gegen die Prüfmannschaft gespielt hat, überschreibt das Genom des anderen mit einer mutierten Kopie seines eigenen.

Physikengine. Da diese selbst nur sequentiell läuft, wurde auf einer höheren Stufe parallelisiert. Pro Ausführungsfaden existiert eine eigene Physiksimulation und somit eine Evaluierungsumgebung für 2 Schwärme (ein „Stadion“). In einem ersten Schritt wurde die Evolution als Steady-State-Algorithmus implementiert. Steady-State bedeutet, dass es keine Generationen gibt, sondern ständig einzelne Individuen aus der Population entfernt und durch Nachkommen ersetzt werden [Whi89]. Die Threads führen die Evolution komplett parallel aus. Bei einer Population von 80 Schwärmen sucht sich jeder Thread zwei Schwärme aus, die nicht gesperrt sind und sperrt sie selbst über ein Mutex. Jetzt lässt er die beiden Schwärme 4000 Zyklen lang in seiner Simulation gegeneinander spielen und ersetzt am Ende den Verlierer des Spiels durch eine mutierte Kopie des Gewinners. Dann gibt er die Sperren der beiden Schwärme frei und sucht sich 2 neue Schwärme. Mit diesem Verfahren sind bei 4 Threads immer maximal 8 Schwärme gesperrt. Der Durchsatz ist sehr hoch, da die Threads kaum Synchronisationsaufwand und Wartezeiten haben.

Bei dem beschriebenen parallelen Steady-State-Algorithmus kann es passieren, dass ein Thread länger braucht als andere und dadurch 2 Schwärme lange aus der Evolution herausgenommen werden. Dies kann zu Verzerrungen führen, weshalb in einem zweiten Schritt ein Generationenkonzept eingeführt wurde. Hier entfallen die expliziten Sperren (Mutex), stattdessen

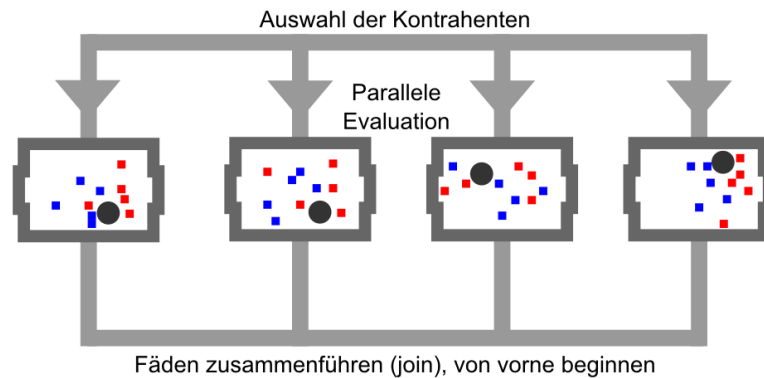


Abbildung 16: Parallelisierung auf Evaluationsebene. Ein kurzer sequentieller Programmteil zieht je 2 Schwärme pro Thread aus der Population ohne Zurücklegen. Sobald alle Threads ihre Mannschaften erhalten haben, beginnt die parallele Durchführung von Spielen zur Evaluation. Auch das Überschreiben des Verlierers durch die mutierte Kopie des Gewinners findet parallel statt.

werden je 2 Schwärme pro Thread durch Ziehen ohne Zurücklegen in einem kurzen sequentiellen Programmteil verteilt (siehe Abbildung 16). Jetzt laufen alle Threads parallel und evaluieren die ihnen zugewiesenen Schwärme. Erst sobald alle Threads ihre Evaluierung abgeschlossen haben, erhält jeder Thread 2 neue Schwärme. Wenn die Menge, aus der die Schwärme gezogen wurden, leer ist, werden alle wieder in die Menge zurückgelegt und die nächste Generation beginnt.

6.4 Fitnessentwicklung

Die folgenden Messungen können zeigen, ob eine darwinsche Evolution im Sinne einer Verbesserung des Verhaltens stattfindet. Dabei wird die gesamte Population, also alle Schwärme, gegen einen sich nicht bewegenden Gegner getestet. Dies gibt nur einen groben Anhaltspunkt über die Leistungsfähigkeit des Schwarms, aber es misst eine ganze Reihe an Grundkompetenzen relativ objektiv. Um eine gute Bewertung zu erreichen, muss der Schwarm den Ball finden, ihn unter Kontrolle bringen und bis ins gegnerische Tor bewegen. Danach wird der Ball wieder in die Mitte versetzt und der Schwarm, der sich oft noch vor dem gegnerischen Tor aufhält, muss sich reorientieren und sich wieder hinter den Ball bewegen, ohne ihn dabei ins eigene Tor zu stoßen. Es wurden 3 Werte gemessen: Die beste erreichte Fitness der Generation gibt eine brauchbare Abschätzung der Leistungsfähigkeit, ist aber zu sehr vom Zufall abhängig („Glück“). Die durchschnittliche Fitness der Gesamtpopulation ist nur mäßig aussagekräftig, da hier die mutierten Nachkommen den Wert stark nach unten ziehen können. Die durchschnittliche

Fitness der besten 25% ist ein guter Kompromiss. Zufallseffekte werden hier stark abgeschwächt und aufgrund des Selektionsverfahrens, das pro Generation die Hälfte der Individuen löscht, ist ein Wert von 25% stark genug eingegrenzt um nur wenige durch Mutation beschädigte Individuen zu umfassen. Der 25%-Messwert dient somit als wichtigster Wert bei dem Vergleich der Messungen untereinander.

Die erste Messung findet unter besonderen Bedingungen statt. Hier ist die Evolution so modifiziert, dass nicht die bessere Mannschaft die schlechtere überschreibt, sondern es ist Zufall, welche der beiden Mannschaften die andere überschreibt. Diese Messung dient als Vergleichswert, bei ihr kann keine darwinsche Evolution stattfinden, sondern nur eine zufällige Drift.

Alle Messungen im Überblick:

- onepop80i1g500rand: Populationsgröße 80, eine monolithische Population, 500 Generationen, keine darwinsche Evolution, sondern zufällige Drift.
- onepop80i1g500: Wie zuvor, bloß mit darwinscher Evolution.
- onepop80i1g500basicsensors: Einfache Sensorausstattung.
- onepop80i1g500basicnocolor: Einfache Sensoren, keine Glühwürmchen-Kommunikation.
- onepop80i4g500: Inselmodell mit Aufteilung der Population in 4 Inseln mit je 20 Individuen.
- twopop80i1g500: Zwei Populationen, die eine verwendet jeweils Individuen aus der anderen als Prüfstein zur Evaluierung der eigenen Individuen. Gemessen wird nur eine der beiden Population. Die Generationszahl bezieht sich ebenfalls nur auf diese eine Population, sodass die Generationen mit den anderen Messungen vergleichbar sind.

6.5 Ergebnisse

Abbildung 17 zeigt die durchschnittlichen Verläufe der Fitness der einzelnen Parametereinstellungen, gemittelt über 25 Läufe. Zur Fitnessbewertung werden jeweils die Durchschnitte der besten 25% einer Generation herangezogen. Die zufällige Selektion bleibt unten nahe 0, dass sie über 0 ist, liegt nur daran, dass die besten 25% herangezogen wurden. Hieran lässt sich erkennen, dass eine zufällige Drift nicht zu einer Verbesserung des Verhaltens führt. Alle anderen Linien steigen an, es ist also in jedem Fall darwinsche Evolution im Gange. Der Zwei-Populationen-Algorithmus enttäuscht, er nimmt

weit abgeschlagen den letzten Platz ein. Alle anderen Parametereinstellungen erreichen am Ende sehr genau dieselbe Bewertung, unterscheiden sich aber im Verlauf davor. Besonders schnell steigt die Linie der einfachsten Sensorausrüstung (onpop80i1g500basicnocolor). Hier profitiert die Evolution offenbar von der Verkleinerung der Dimensionalität des Suchraumes, da nur wenige Sensoren zur Verfügung stehen und die neuronalen Netze somit anfangs automatisch eine geringere Anzahl an Gewichten haben. Auf Platz zwei folgt allerdings bereits die Linie der maximalen Sensorausstattung (onpop80i1g500). Die zusätzlichen Sensoren scheinen die Erhöhung der Dimensionalität des Suchraums nach einiger Zeit ausgleichen zu können. Die Linie der einfachen Sensoren + Glühwürmchenkommunikation schneidet schlecht ab und erreicht das Niveau der Linie ohne Glühwürmchenkommunikation erst später. Das Inselmodell steigt auch erst langsam an, hier liegt der Grund aber unter Umständen darin, dass die Inseln gute Strategien nur langsam zu anderen Inseln weiterleiten, diese sich also nicht so schnell weitreichend etablieren. Dies muss kein Nachteil sein, sondern kann die Diversität langfristig erhalten. Das Inselmodell erreicht trotzdem am Ende die gleiche Fitness wie die anderen Linien.

Eine Ergänzung zu den Messkurven sind die Histogramme aus Abbildung 18. Sie zeigen eine Statistik, wieviele Tore die 25 Läufe pro Parametereinstellung maximal erzielt haben (über alle Generationen hinweg, gemessen sind die durchschnittlichen Tore der besten 25%). Die zufällige Selektion kommt hier auf 0 Tore, der Zwei-Populationen-Algorithmus erreicht oft auch nur 0 Tore, nur einzelne Läufe sind hier erfolgreich. Die einfache Sensorausstattung mit Glühwürmchen-Kommunikation (basicsensors) schneidet hier besser ab als die ohne Glühwürmchen-Kommunikation (basicnocolor). Diese und die Sensorvollausstattung sind die einzigen Parametereinstellungen, die in manchen Läufen 3 oder 4 Tore als Durchschnitt der besten 25% erreichen. Letztendlich ist die Stichprobe mit nur 25 Läufen zu gering um sichere Schlüsse zu ziehen. Die aktuellen Messwerte legen jedoch den Schluss nahe, dass mehr Sensoren auch bessere Ergebnisse erzielen können, dies jedoch nur in wenigen Läufen erreicht wird.

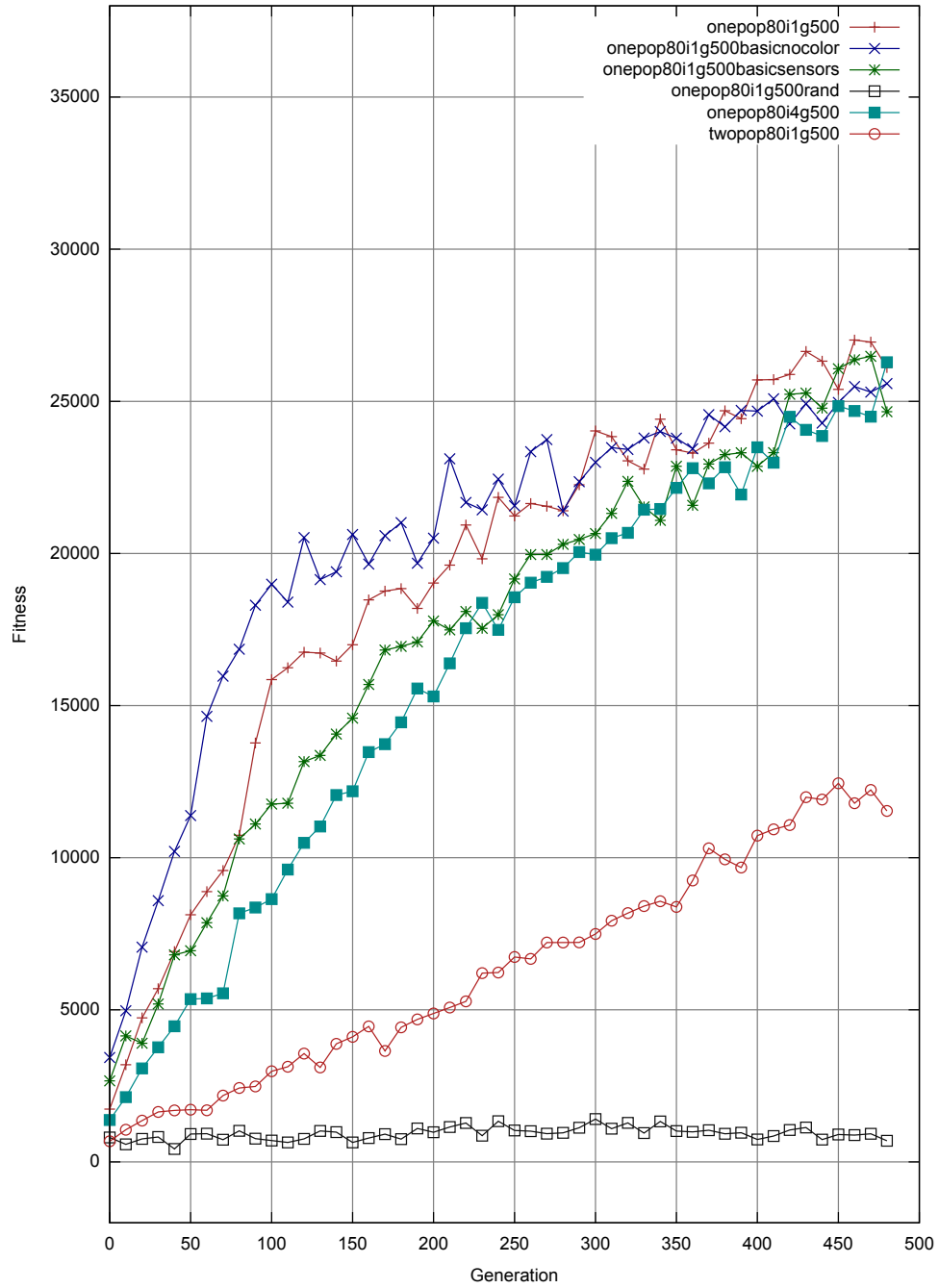


Abbildung 17: Fitnessentwicklungen bei verschiedenen Parametern. Jeder Datenpunkt steht für die Fitness der besten 25% in der Generation, gemittelt über 25 Evolutionsläufe.

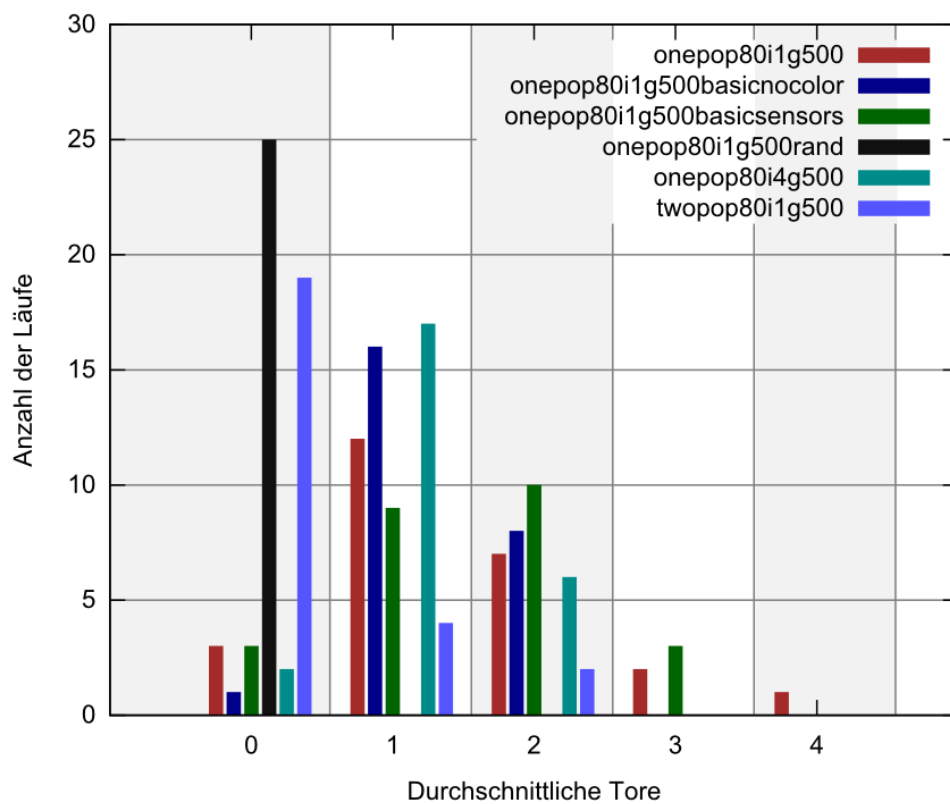


Abbildung 18: Histogramme der Parametervarianten im Vergleich.

7 Fazit und Ausblick

7.1 Zusammenfassung

In dieser Studienarbeit wurden neuronale Netze als Steuerungsprogramme für Roboter verwendet, die im Schwarm kooperativ putzen und Fußball spielen. Die neuronalen Netze wurde dabei durch einen zentralen evolutionären Algorithmus mit globalen Informationen trainiert. Die Roboter selbst agierten aber dezentral nur mit lokalen Informationen.

Dabei wurde darwinsche Evolution im Sinne einer iterativen Verbesserung des Verhaltens beobachtet. Sowohl im Putz- als auch im Fußballscenario dominierten einfache Strategien, die wahrscheinlich lokale Optima darstellen. Im Falle des Putzscenario ist dies das kreisende Wall-Following, im Fußballscenario sind es Strategien, die die Roboter hinter den Ball führen und ihn in Richtung gegnerisches Tor bewegen. Das Fußballscenario bietet noch Raum für Verbesserung, z.B. durch Kommunikation zwischen den Teammitgliedern. Das Prinzip, Schwärme von Robotern mit identischen neuronalen Netzen auszustatten und so einen Schwarm als Individuum in der Evolution zu betrachten, hat funktioniert. Es benötigt aber eine sehr lange Evolutionszeit, da jede Evaluierung eines Schwarms einen kompletten Simulationslauf über eine ausreichend hohe Anzahl an Zyklen erfordert.

7.2 Ausblick

In dieser Arbeit wurde eine direkte Kodierung von Genom zu neuronalem Netz verwendet. Jede einzelne Synapse ist direkt im Genom kodiert. In der Natur stellt die DNA keine direkte Beschreibung des Organismus dar. Der Organismus emergiert stattdessen aus dem Zusammenspiel vieler Zellen, die durch Aktivierung und Deaktivierung von DNA-Teilen unterschiedliche Rollen wahrnehmen. Der Ausgangspunkt des Organismus liegt dabei in einer einzelnen Zelle, die sich teilt. Die neue entstandenen Zellen teilen sich ebenfalls weiter und beginnen sich auszudifferenzieren. Chemische Gradienten entlang mehrerer Achsen im Organismus teilen der Zelle ihren Aufenthaltsort und somit ihre Rolle mit. Dieses Grundprinzip wurde bereits von Gruau auf künstliche neuronale Netze übertragen. Die neuronalen Netze entstehen hierbei indirekt aus einer zellulären Kodierung [GIID⁺94]. Gauci und Stanley beschäftigen sich mit der Frage, ob eine Entwicklung, wie die eines Embryos zum vollständigen Organismus, für eine indirekte Kodierung erforderlich ist. Mit HyperNEAT [GS07] stellen sie ein indirekte Kodierung für neuronale Netze vor, die ohne Entwicklung auskommt. Dabei wird das neuronale Netz durch ein anderes neuronales Netz erzeugt. König und Schmeck [KS09] verfolgen einen Ansatz, der das Genom durch einen erzeugenden

endlichen Automaten in den resultierenden endlichen Automaten übersetzt.

In dieser Arbeit wurden vor allem zentrale evolutionäre Algorithmen zum Training der neuronalen Netze verwendet. Während der Implementierung wurde allerdings auch dezentrale Evolution getestet. Dabei wurde erfolgreich ein Kollisionsvermeidungs-Verhalten sowie eine einfache Strategie zum Putzen entwickelt. Die dezentrale Evolution hat verschiedene Vorteile gegenüber dem in dieser Studienarbeit verwendeten Ansatz. Insbesondere bei Experimenten mit echten Robotern, die selbst lernen sollen, ist ein dezentrales Lernen wünschenswert.

Literatur

- [Cul96] Joseph C. Culberson. On the futility of blind search. Technical report, EVOLUTIONARY COMPUTATION, 1996.
- [Cyb89] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, December 1989.
- [D.00] Nolfi S. & Floreano D. Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines. 2000.
- [DJW97] Stefan Droste, Thomas Jansen, and Ingo Wegener. Optimization with randomized search heuristics – the (a)nfl theorem, realistic scenarios, and difficult functions. *THEORETICAL COMPUTER SCIENCE*, 287:2002, 1997.
- [DK79] R. Dawkins and J. R. Krebs. Arms Races between and within Species. *Royal Society of London Proceedings Series B*, 205:489–511, September 1979.
- [Fah90] Scott E. Fahlman. The recurrent cascade-correlation architecture. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 3*, pages 190–196. Morgan Kaufmann Publishers Inc., 1990.
- [FL90] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann, 1990.
- [GIID⁺94] Frédéric Gruau, L’universite Claude Bernard lyon I, Of A Diplome De Doctorat, M. Jacques Demongeot, Examinators M. Michel Cosnard, M. Jacques Mazoyer, M. Pierre Peretto, and M. Darell Whitley. Neural network synthesis using cellular encoding and the genetic algorithm. 1994.
- [GS07] Jason Gauci and Kenneth Stanley. Generating large-scale neural networks through discovering geometric regularities. pages 997–1004, 2007.
- [Hil90] W. Daniel Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Phys. D*, 42(1-3):228–234, 1990.
- [Hyo96] Heikki Hyotyniemi. Turing machines are recurrent neural networks. 1996.

- [JP96] Hugues Jüille and Jordan B. Pollack. Co-evolving intertwined spirals. pages 461–468, 1996.
- [KNA07] Nadav Kashtan, Elad Noor, and Uri Alon. Varying environments can speed up evolution. *Proceedings of the National Academy of Sciences*, 104(34):13711–13716, August 2007.
- [KS08] Lukas König and Hartmut Schmeck. Evolving collision avoidance on autonomous robots. *Biologically-Inspired Collaborative Computing*, pages 85–94, 2008.
- [KS09] Lukas König and Hartmut Schmeck. A completely evolvable genotype-phenotype mapping for evolutionary robotics. pages 175–185, September 2009.
- [MC06] Thomas Miconi and Alastair Channon. The n-strikes-out algorithm: A steady-state algorithm for coevolution. pages 1639–1646, 2006.
- [Mic07] Thomas Miconi. The road to everywhere: Evolution, complexity and progress in natural and artificial systems, 2007.
- [Mic09] Thomas Miconi. Why coevolution doesn't "work": Superiority and progress in coevolution. 5481:49–60, 2009.
- [RHW88] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. pages 696–699, 1988.
- [Ros88] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. pages 89–114, 1988.
- [SM02] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, (AI2001-290):99–127, 2002.
- [SM04] Kenneth O. Stanley and Risto Miikkulainen. Competitive coevolution through evolutionary complexification. (AI2002-298):63–100, 2004.
- [vV73] L. van Valen. A new evolutionary law. *Evolutionary Theory*, 1:1–30, 1973.
- [Whi89] Darrell Whitley. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. pages 116–121, 1989.

- [WM97] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, 1(1):67–82, 1997.
- [Wol83] Stephen Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55(3):601–644, Jul 1983.

8 Anhang: Spielszenen

Am besten lässt sich das Verhalten der Roboter in bewegter Form beurteilen (siehe beiliegende CD-ROM). Die folgenden Trajektorienbilder können im Zusammenspiel mit der Text-Beschreibung aber einen guten Überblick über verschiedene Verhaltensweisen bieten. Bei vielen Bildern ist der Ball in der Mitte, da er nach einem Tor dorthin zurück versetzt wurde. Tore sind dort aufgetreten, wo die schwarze Linie endet und in einem farbigen Kreis mündet. Der farbige Kreis zeigt an, für wen das Tor ein Vorteil war (aber nicht unbedingt, wer es geschossen hat). Das blaue Team spielt auf das rechte Tor, das rote auf das linke Tor.

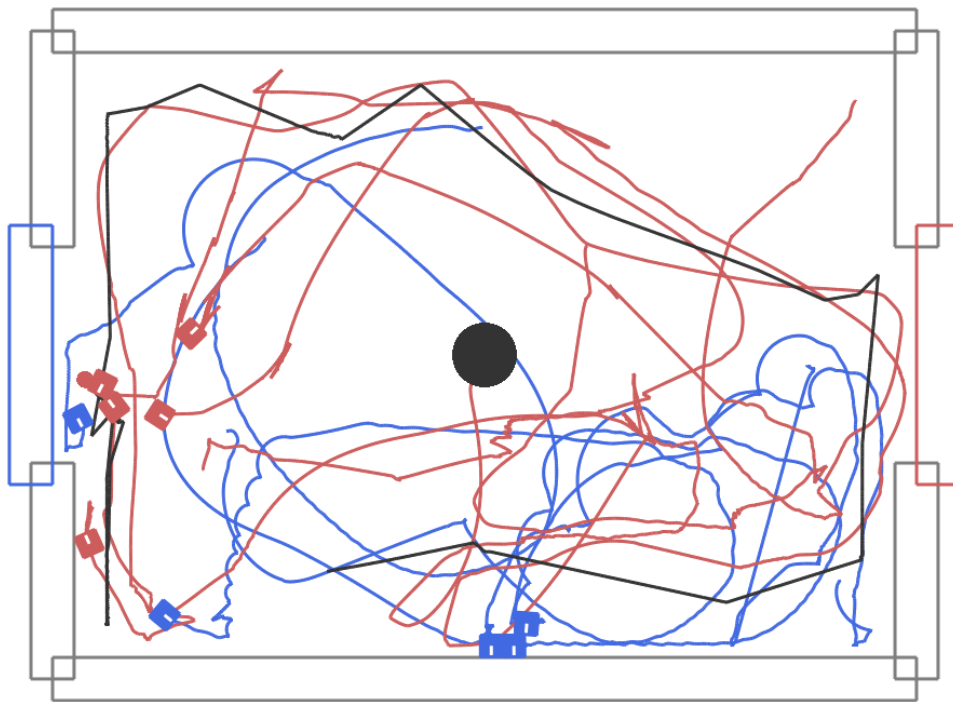


Abbildung 19: Tor verhindert und selbst Tor geschossen (knapp 400 Generationen): Der Ball beginnt unten, wird vom blauen Team nach rechts getragen, Rot verhindert aber den Torschuss, übernimmt den Ball und transportiert ihn oben entlang nach links bis vors blaue Tor. Ein blauer Spieler geht ins Tor hinein und verhindert die erste rote Torchance, indem er den Ball abblockt. Der Ball prallt an der unteren Wand ab und geht wieder nach oben, verfolgt von 2 roten Spielern, die ihn zusammen an Blau vorbei ins Tor drücken.

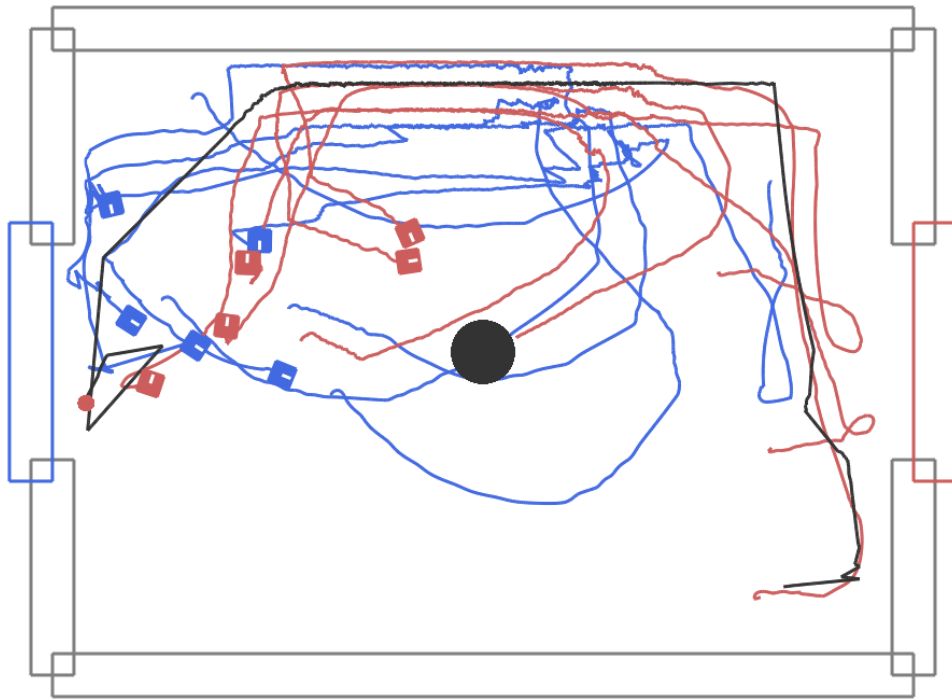


Abbildung 20: Nachschuss (knapp 400 Generationen): Der Ball startet rechts unten, wird von einem nahen roten Spieler nach oben geschoben, zwei weitere rote Spieler schieben auch mit. Am oberen Rand fangen die blauen Spieler den Ball ab und drücken von links und unten dagegen. Die Überzahl der roten Spieler schiebt den Ball aber langsam (zittrige Linien) weiter nach links und löst ihn schließlich von der Wand. Vor dem blauen Tor geht der Ball zuerst vorbei und prallt unten am Eck ab. Im Nachschuss drückt der immer noch nahegelegene rote Spieler den Ball aber ins Tor.



Abbildung 21: Einkesseln (knapp 1000 Generationen): Beide Mannschaften verfolgen den Ball und versuchen, sich hinter ihm zu platzieren, um ihn in Richtung gegnerisches Tor anzustoßen oder zu schieben. Dabei kann es insbesondere an Wänden oder in Ecken vorkommen, dass der Ball umschlossen wird. In manchen Fällen lösen sich solche Situationen wieder auf, weil die Umklammerung asymmetrisch wird und der Ball durch eine Lücke herausgeschoben wird.

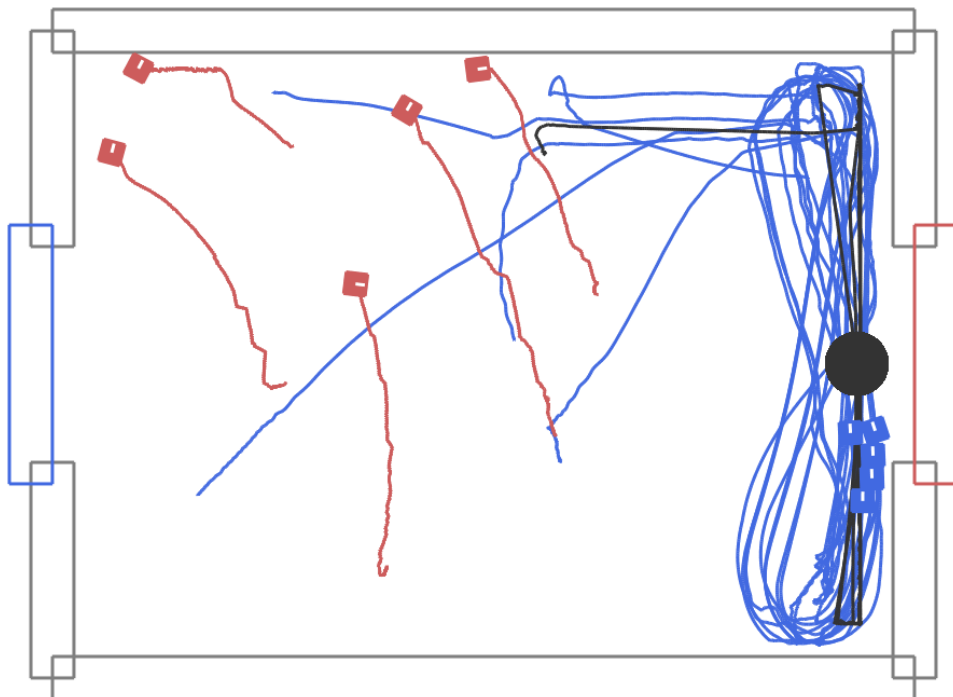


Abbildung 22: Das rote Team hält sich heraus, während das blaue Team den Ball endlos in der Vertikalen vor dem roten Tor rauf und runter bewegt. Beim blauen Team fehlt die Kommunikation, es müsste ein blauer Spieler vor dem Tor stehen bleiben und auf den Ball warten. Die Sensoren der Roboter nehmen die anderen Roboter nicht gut genug wahr.

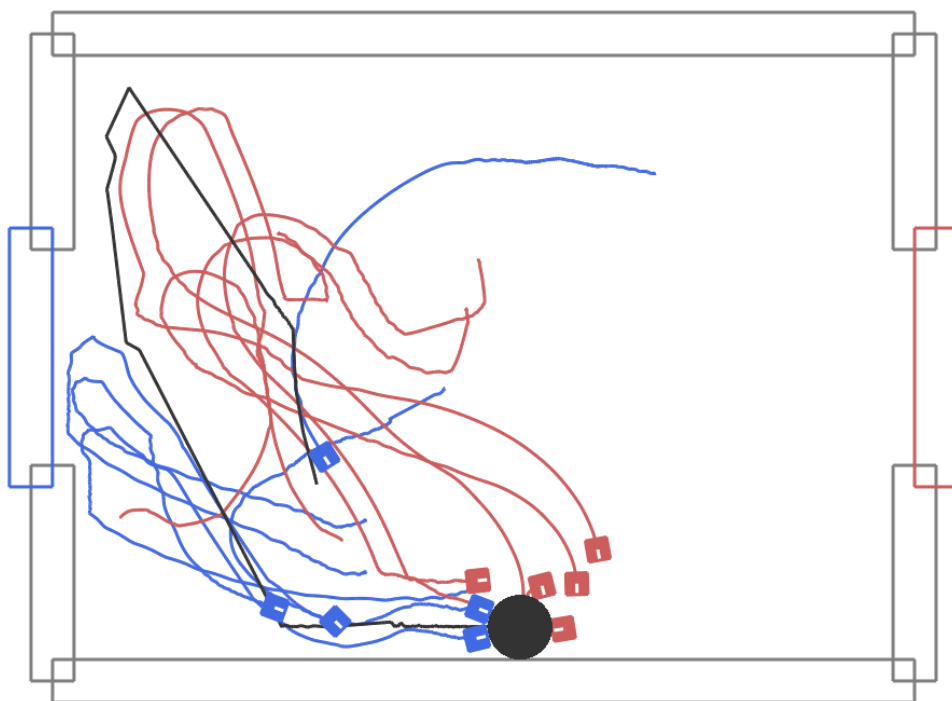


Abbildung 23: Die Roboter fahren Kurven, um sich hinter dem Ball zu platzieren und ihn in Richtung gegnerisches Tor zu bewegen.



Abbildung 24: Der Ball startet im mittleren unteren Bereich und wird von einem blauen Roboter nach rechts bewegt. Am ersten starken Knick der schwarzen Linie hat ein roter Roboter den Ball gerammt. Letztendlich schafft das blaue Team es, den Ball in der roten Hälfte zu halten und schräg ins rote Tor zu bewegen. Der Ball wird in die Mitte zurück versetzt und die Roboter beider Teams richten sich nach ihm aus und fahren auf ihn zu.